108

AD A030162

UNITED STATES AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

(9) [illegible]

(6)

AN EXPLORATORY STUDY OF SOFTWARE COST ESTIMATING AT THE ELECTRONIC SYSTEMS DIVISION.

THESIS

(14) GSM/SM/76S-4

(10) Thomas J. Devenny
Capt USAF

(12) [illegible]

(11) [illegible]

Approved for public release; distribution unlimited.

AN EXPLORATORY STUDY OF SOFTWARE COST ESTIMATING AT THE ELECTRONIC SYSTEMS DIVISION

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

by

Thomas J. Devonny, B.S.
Capt USAF
Graduate Systems Management

July 1976

Approved for public release; distribution unlimited.

Preface

Should this research effort have any positive effect upon the acquisition of weapon system software, the thanks are due primarily to the many engineers and managers at ESD who fully supported this research. Providing both favorable and unfavorable information on a sensitive topic, their aim was to improve software management and not to hide problems or failures in estimating software cost.

Additional thanks are due to Col. W. Woodruff, Director of the SATIN IV program office, for providing the TDY funds essential for this research. Thanks also to Lt Col Art Roscoe for the initial idea which led to this research. The end result has been interesting, rewarding, and hopefully useful.

Maintaining objectivity and producing a readable report were difficult tasks for this writer. For their support in these areas, thanks are due to Dr. C. McNichols and Dr. R. Tripp, my thesis adviser and reader.

Editing, proofreading, typing, and re-typing are some of the less rewarding tasks associated with any research. Special thanks to my wife Betty, who did all these jobs for a price still being negotiated.

Contents

List Of Figures

List Of Tables

Abstract

The estimate of software development cost is a key piece of information in many software management decisions. No technique exists which can consistently produce the reliable and accurate cost estimates which managers need. This thesis research effort explored the software cost estimating process at the Electronic Systems Division (ESD) of the Air Force Systems Command. The purpose of the research was to provide managers, researchers, and cost estimators with a better insight into the cost estimating process.

Data were gathered from 16 major software acquisitions at ESD using both a structured interview and contractor furnished Cost Performance Reports. The research findings identified some major problems which are currently inhibiting the development of accurate and reliable software cost estimates.

To reduce these problems, recommendations are made to adopt a common cost estimating technique and to modify the use of contractor furnished software cost information. While the research was limited to ESD, the research findings and the recommendations may be applicable to other DoD software acquisition agencies.

AN EXPLORATORY STUDY OF SOFTWARE COST ESTIMATION AT THE ELECTRONIC SYSTEMS DIVISION

I. Introduction

DoD program managers will buy an estimated three billion dollars worth of software in 1976 (Aviation Week, 1976:41). Other managers throughout the United States will buy an additional fifteen billion dollars worth of software (Boehm, 1975:4). These managers face a common problem in predicting or estimating the cost of software.

In general, the accuracy of software cost estimates has been poor. Underestimation by a factor of 2.5 is common while overestimations are unheard of (Schwartz, 1975:56). While a number of efforts have been made to develop improved cost estimation techniques, no generally accurate nor reliable method for estimating software development costs has been found (Morin, 1974).

Objective

The objective of this research effort is to describe the nature and status of the software cost estimating environment.

Purpose

The purpose of this research is to provide managers, researchers, and cost estimators a better insight into the software cost estimating problem. It is hoped that this increased insight into the problem area will ultimately result in improved software cost estimation.

Scope and Limitations

This research effort is limited to describing the software cost estimating environment in the Electronic Systems Division (ESD) of the Air Force Systems Command. ESD is located at Hanscom AFB in Massachusetts and is responsible for a major portion of Air Force software acquisitions. While software is also procured at other Air Force locations, the ten weeks available for this research effort dictated that only the ESD environment could be explored.

In addition, the research is limited to those major software acquisitions at ESD which are currently proposed, are under way, or are recently completed. This limitation was necessary due to the general non-availability of data concerning past programs and limited the data sources to twenty-one major software acquisitions.

No classified data was collected during the research effort. However, this had no impact upon the research due to the unclassified nature of the data which was sought.

One last limitation on this research effort is caused by the methodology which was selected. The data were largely collected using a structured interview. The limitations of this technique are discussed in Chapter 4, Methodology.

Applicability of Research Findings

Since the research is limited only to software developments at ESD, the applicability of the research findings to other software developments may reasonably be questioned.

In many ways, the ESD environment is unique. ESD receives systems engineering support from the MITRE Corporation, a Federal Contract Research Center (FCRC). The relationship between ESD program offices and MITRE is somewhat unique in the DoD. Also, the types of systems procured by ESD are primarily command, control, and communications systems. The relationship between these types of software developments with other types of software developments (e.g. avionics, inventory control, personnel systems) is not well defined.

However, despite the somewhat unique aspects of the ESD environment, there are many facets of the environment which are quite similar to that of other DoD software acquisition agencies. The types of contracts issued, the regulations covering system acquisition, and the common weapon system cycle tend to make the software acquisitions of ESD similar to that of other DoD agencies. Also, the software systems at ESD are generally large software systems in excess of $500,000. The very size of the software acquisitions tends to produce many common effects in software development regardless of the intended application of the software system.

Therefore, while the uniqueness of the ESD environment may tend to limit the application of the research findings, it appears reasonable to assume that these findings may apply in a general fashion to other large software developments made by other DoD agencies.

Assumptions

Since the research is descriptive in nature, no explicit assumptions are made concerning the software cost estimating environment at ESD.

Utility

To date, most of the research in the area of software cost estimation has been in the search for causative relationships. Researchers are trying to determine the relationship between various software parameters and cost. There have been no published efforts which have described the environment in which cost estimates are made and used. The writer believes that such a research effort has value to three types of people.

The Manager. The manager of a software development makes a number of decisions which rely on a software cost estimate. To make these decisions, a manager must understand the data with which he is presented. This research seeks to aid the manager's understanding of the software cost estimating process so that he may better utilize the estimates presented to him.

Managers are frequently evaluated on how well budgeted cost agrees with actual cost. If a software project experiences a 200% overrun, the manager's performance may look poor. However, with the current state of software cost estimating, it is difficult to determine whether the 200% overrun was due to poor management or poor estimating. This research effort hopes to assist managers in understanding why software cost estimates are difficult to prepare and frequently erroneous.

The Researcher. Research continues in an effort to establish valid cost estimating relationships for software. To date, these efforts have met with very limited success. A major objective of this descriptive research is to provide other researchers with clues to cause and effect relationships which may exist. This effort may help to explain how and why certain cost estimating techniques are used or not used. Equipped with a better insight to the nature of the software cost estimating environment, other researchers may be assisted in their efforts.

The Cost Estimator. The cost estimator is faced with the task of assigning a cost to software, a very intangible object. Other cost estimators need only to measure the weight and speed of an aircraft to reasonably estimate aircraft costs. However, the software cost estimator does not find his job easy. He is faced with numerous techniques which require the use of a large number of vaguely defined variables. He continues to search for a better method, but cannot amass enough experience on his own to determine which techniques are better than others or which techniques are easier to apply.

This research effort does not develop a new cost estimating technique. It does not determine causative relationships. Instead, this effort only seeks to describe the environment surrounding the software cost estimation process. A basic objective is to communicate the experiences of some software cost estimators to other software cost estimators. The sharing of experience in this area has been limited. This study seeks to enhance the sharing of information.

Thesis Outline

The thesis consists of six chapters. The first chapter has discussed the nature of the research effort.

Decision Making and Software Cost Estimates. Chapter 2 discusses the decisions made during the weapon system acquisition process which rely upon a software cost estimate. These decisions range from selecting a weapon system to awarding software contracts.

State of the Art of Software Cost Estimating. Once the reader is familiar with the decisions which utilize a software cost estimate, Chapter 3 examines the current state of the art of software cost estimating. To understand the ESD environment, one must first understand the state of the art of software cost estimating as presented in the literature. Five general categories of cost estimating techniques are reviewed. The problems associated with using these techniques or with developing new ones are also discussed.

Methodology. After the background provided by Chapters 2 and 3, Chapter 4 details the methodology used for the research effort. The hypotheses which were initially formed as well as the interview developed to test the hypotheses are discussed in general.

Findings. Chapter 5 contains the research findings. Each of the initial hypotheses are reviewed along with the data which was collected. The data are analyzed and a determination is made concerning the research hypotheses.

Recommendations. Chapter 6 closes the research effort with some recommendations based on the findings.

II. Decision Making And Software Cost Estimates

Before looking at how software cost estimates are made, it is important to understand how the estimate is used in making decisions. This chapter discusses the decisions made during the weapon system acquisition process which are influenced by the software cost estimate.

Economic Evaluation Of Weapon Systems

The software development cost estimate is used in the economic evaluation of weapon system alternatives. For example, the Air Force might be faced with deciding whether to update an aging air defense system or to develop a new, advanced system. To make this decision, decision makers must understand the cost of the hardware and software associated with each alternative.

While software cost may have been minor in the past, it is now frequently a major component of total system cost. For example, the World Wide Military Command and Control System (WWMCCS) will expend over $722 million on software and less than $100 million on hardware (Air Force OSR, 1973:28). During nine years of development, the Army spent an estimated $467 million on software development for the Safeguard System (Ashe et al, 1975:2-15). Over a four year period, the Minuteman program office spent $124 million on software (Ashe et al, 1975:2-22). In the 1980's software can be expected to become an ever increasing portion of weapon system cost. Therefore, to evaluate alternative weapon systems, a decision maker must have a reliable software cost estimating technique.

Design Tradeoffs

The software cost estimate is also used in making engineering design tradeoffs within a weapon system. Design engineers must decide whether to select an older computer with a large set of support software, or to select a newer computer with more computational speed but less support software. They must decide whether a particular function is best performed by software, by hardware, by firmware, or even by manual procedures. They must also decide whether to "make or buy" software.

With the advent of microprocessing, "off the shelf" software packages, and other developments, the design engineer is faced with a myriad of design alternatives. To select the best alternative, the engineer must be able to reliably estimate the software development cost associated with each alternative.

Budgeting

The software cost estimate forms the basis for the budget or financial plan of the weapon system. Errors in the estimate cause errors in the budget. If software costs are seriously underestimated, a program manager might have to request additional funds from DoD or Congress. Additional funds might not be readily available causing the weapon system to slip its schedule while awaiting additional funds.

In general, the request for additional funds is quickly labeled "overrun" and not "inaccurate estimate". The program manager finds that his program is subjected to increased surveillance and that his management flexibility is reduced (Large, 1974:10).

Therefore, to avoid the problems associated with seeking additional funds and to avoid the criticism associated with overruns, a program manager needs a reliable technique to estimate and budget software development cost.

Competitive Contract Award

The software cost estimate is also a primary factor in the award of software development contracts. Proposals from competing contractors are analyzed and the contract is usually awarded to the lowest bidder. If the lowest bidder's price is unreasonably low, two problems are created.

First, the contract award is inequitable. Awarding to a bidder who has seriously underestimated the cost of a contract and is eventually bailed out by the government penalizes other bidders who more accurately understood and estimated the contract requirements.

A second problem develops from awarding a contract at an unreasonably low cost. In a study done by RAND Corporation, their findings indicated that underestimating a contract leads to cost growth. An excessively low bid forces a contractor to make unwise decisions in an attempt to stay within costs. For example, the contractor might award to a less qualified sub-contractor or place insufficient emphasis on the production, operation, and maintenance aspects of a weapon system. When contract costs rise, the government frequently bails out the contractor and is faced with paying not only the true or expected cost of the contract, but also the additional costs caused by the contractor's unwise decisions (Large, 1974).

Therefore, in order to equitably award a contract at a reasonable price, contractors and contracting officers need a reliable technique of estimating software costs.

Contract Schedule and Manpower

The winning contractor uses his estimate of software development cost to determine how many men to assign to a project and how many months to allocate to software development. If software costs are underestimated, either too few programmers will be assigned to the project or too short a period will be allocated for software development.

When the contractor becomes aware of the true size and cost of the software, he must either add programmers to the project or extend the project's schedule. Either of these actions can create new problems. For example, when new programmers are added to a project, the original programmers stop producing software while they train the new programmers and develop a new software production plan. The cost of this additional training and planning effort increases software cost even higher. Maintaining the existing schedule by adding people might appear simple, but in many cases the results are disastrous (Brooks, 1975:24).

On the other hand, extending the project's schedule also creates problems. Most weapon systems are both hardware and software developments with software frequently on the critical path of the schedule. While waiting for a small software effort to be completed, an entire multi-million dollar program might be delayed. The indirect cost of software due to this type of delay could be fifty times the additional software cost (Findley, 1974:15). This large indirect cost is due to the significant impact software can have upon the schedule or performance of a weapon system.

Another less obvious problem can result from providing inadequate time or manpower to a software project. When the software project manager finds himself running short of time and manpower, one possible action he might take is to reduce the quality of the software product. He might encourage inefficient programming, less testing or poorer documentation in an effort to remain within his inadequate resources.

Therefore, to avoid the problems associated with inadequate schedules and manpower, a manager needs a reliable technique for estimating software development resources.

Contract Status

Another use of the software cost estimate is in monitoring the status of software development. It is difficult to establish clearly defined technical milestones for software. Because of this difficulty, a common method of evaluating the technical status of software development is to compare the contractor's actual cost to the contractor's budgeted cost. For example, if a contractor has expended 90% of his budgeted cost for software, one might infer that the software is 90% complete. However, such an inference is only correct if the budgeted cost of software is accurate. If the budgeted or estimated cost of software is wrong, then the government is only monitoring financial expenditures and not technical progress.

If valid status indications are to be obtained for software development, an improved set of technical milestones for software as well as a reliable software cost estimating technique are required.

Reasonable Estimates And Good Decisions

Cost estimators cannot predict the future. No crystal balls are involved. Instead, a cost estimator tries to determine a "reasonable" value for the resources needed to carry out a particular plan. Therefore, a decision maker cannot expect highly accurate cost estimates to support his decisions.

However, all of the previously discussed decisions do not require highly accurate estimates. Most of these decisions could probably be made reasonably well with an estimate that varied as much as plus or minus 25%. However, as we shall see in later chapters, current techniques available for estimating software cost frequently have errors in excess of 250%. With errors of this magnitude, making good decisions concerning software developments becomes increasingly difficult if not impossible.

If we want to make good decisions concerning software, we need to be able to reasonably estimate software costs. If we want to make tradeoffs between systems and within a system, if we want to award contracts equitably and carry them out successfully, then decision makers must have a reliable technique for estimating software development cost. With these decisions in mind, we can now look at the state of the art of software cost estimating and examine the techniques which are currently available.

III. State Of The Art Of Software Cost Estimating

Having seen how managers and engineers use a software cost estimate in making decisions, we can now examine the various techniques which are available for making software cost estimates. These techniques can be categorized into five types: unit price, specific analogy, expert opinion, cost to cost relationship, and non-cost to cost relationship. Each of these techniques seeks to relate an historical cost to a future cost (Jones, 1965:19-20). This chapter examines the state of the art of these five techniques. The problems associated with using them or with developing a new technique are also discussed.

Unit Price Technique

Probably the most common estimating technique used to predict software cost is the unit price or cost per instruction technique. This technique first develops an expected cost for a single computer instruction in a certain computer language. For example, after examining previous JOVIAL computer program developments, an average or expected cost of a JOVIAL computer instruction can be determined. The second step in the technique is then to size the new software by estimating the number of JOVIAL instructions required. Multiplying the expected cost of a JOVIAL instruction by the estimated number of JOVIAL instructions yields the estimated cost of the software development.

The relative simplicity of this technique may account for its frequent use. However, there are numerous problems associated with this technique. First, selecting an appropriate cost per instruction factor is difficult because of the lack of a well defined data base. Second, the technique requires the estimator to predict the number of computer instructions in a computer program. This is sometimes as difficult as predicting software cost. Third, there is some question about the accuracy which results when the number of instructions is used to predict cost. Since this technique is common in use, it is important to understand each of these problem areas.

<u>"Cost" Per Instruction</u>. While the term "cost per instruction" might appear simple and straightforward, it is anything but that. Defining exactly what is meant by "cost" and what cost elements are included is difficult. For example, does the term cost only measure the direct labor hours of computer programmers or does it include the direct labor of managers, keypunchers, secretaries, and computer operators? Is only direct labor included or is the cost of overhead included? For example, computer time might represent as much as 25% of a software development cost (Wolverton, 1974:629). However, some cost per instruction factors might not include such a cost.

In fact, any approach in accumulating software cost elements would probably be reasonable as long as the approach was consistent and well defined. This is a major problem since no guidelines exist which indicate which cost elements should be accumulated in determining a cost per instruction factor. Even if such a guideline existed, contractors might have difficulty in following the guideline due to differences in corporate accounting practices. For example, one company might allocate computer costs based on direct programmer man-hours while another company might allocate these costs based on direct programmer salary dollars. The two techniques would introduce problems in comparing the cost per instruction factors of two companies. Numerous other areas exist where differences in contractors' accounting systems might preclude comparison of data.

However, even if these difficulties could be eliminated, another major problem is encountered in measuring cost per instruction. Software development is a process. The exact beginning and end of the process are not well defined. Some cost per instruction factors only address the cost during the design, coding, and debugging phases of software development. However, a software development has a significant amount of activity prior to and after these phases. For example, before a contractor can begin to design a computer program, he frequently has to analyze the user's operational requirements and prepare a system specification. System interfaces must be defined and data bases designed. In addition, the contractor frequently develops program test plans and specifies input/output message formats. All of these tasks consume resources during a software development prior to the design of a single computer program.

After a programmer has debugged a certain computer program, the software development is usually far from complete. The contractor normally must plan and conduct an integration test of all programs and a system test of hardware and software components. Programmers must sometimes train user personnel or maintain the software for a certain period after the software has been delivered. These costs are usually substantial and they occur after the software has been debugged.

If agreement can be reached on what cost elements should be measured in the cost per instruction, then it is equally important to agree on what parts of the software process are to be measured. Each part of the process needs to be well defined and cannot simply be referred to in broad terms such as "design" or "test". When both the elements of cost and the parts of the software development process have been agreed upon, a common meaning for the term "cost" can be established for determining an appropriate cost per instruction factor.

Cost Per "Instruction". Unfortunately, determining the appropriate cost would not end the problems of the term "cost per instruction". There are considerable variances in the interpretation of the term "instruction". For example, some estimators claim that the instructions which should be counted are source instructions. They reason that a single source statement is the product of the programmer and is the best measure of programmer output. Others, however, claim that the number of object instructions should be measured. Object instructions are the computer instructions which result after the original source instructions have been compiled by the computer. This group believes that the use of object instructions more accurately measures programming output.

While either source or object instructions might be appropriate measures, one cannot use both. A single source instruction in JOVIAL can lead to five or more object instructions. Frequently, literature which discusses cost per instruction factors does not clarify which type of instruction was used in determining the factors.

Another problem exists in identifying the language which is being used. Much of the literature simply refers to the cost per instruction for an HOL (Higher Order Language) and does not differentiate between Fortran, Cobol, or JOVIAL developments. The use of a common HOL estimating factor ignores the differences among languages.

Still another problem is whether or not instructions should be classified by type. A set of instructions in a time sensitive program might be much more difficult to develop than a set in which time was not a factor.

Both the System Development Corporation (SDC) researchers (Nelson, 1967) and the Tecolote researchers (Frederic, 1974) found that by classifying instructions into certain categories, the corrolation between the number of instructions of a category and cost was significantly improved. Yet, despite the differences in categories of instructions, most literature simply refers to a single cost per instruction factor without identifying the categories of instructions which led to the particular cost factor.

Perhaps the least obvious problem in counting the number of instructions produced is to determine which instructions should be counted. Most weapon system software developments are not limited to the development of operational software for the weapon system. Instead the development includes assemblers, compilers, libraries, simulators, test tools, and data reduction programs. Adding these instructions to the instructions from the operating programs yields the total number of deliverable instructions. However, the literature frequently does not identify whether the cost per instruction factor was determined based on the number of operational computer instructions or the number of delivered instructions. Since this difference can be quite substantial (Manley, 1975:53), it is imperative that the basis for counting instructions be as well defined as the basis for defining cost.

It can be seen that the simple phrase "cost per instruction" is not quite that simple. Any effort to accumulate a historical data base would have to require that explicit definitions be established and that accounting systems be similar. Without such an effort, the data used to develop a cost per instruction factor would be questionable. This was demonstrated in a research effort by Tecolote Research. At the start of their research, they had 387 data points from different software development efforts. The data included cost information (sometimes in terms of man-months) and an instruction count. The lack of information concerning the cost elements, the software development phases measured, and the types of instructions which were counted led the Tecolote researchers to abandon 382 of their 387 data points. They then proceeded to develop an estimating methodology based on the five data points about which then had reasonable information (Frederic, 1974).

Estimating The Number Of Instructions. If an appropriate cost per instruction factor has been developed by careful data collection, the estimator then needs to estimate the number of instructions required in the design of a weapon system. Weapon system software packages frequently involve hundreds of thousands of instructions and an accurate estimate is difficult to make. This difficulty is not unique to weapon system software packages. For example, on a software development by UNIVAC for United Air Lines, the initial estimate of the number of instructions required for each transaction was 9,000. The system was cancelled when the number of instructions per transaction had escalated from 9,000 to 146,000 (Schwartz, 1975:56). This type of occurence is all too frequent in both commercial and defense developments.

Two factors which affect the ability of the estimator to predict the number of computer instructions are the experience of the estimator and the amount of detailed design information which is available. To estimate how many instructions are required in a software package, an estimator normally breaks the package down into programs and subprograms until the size of the software modules can be estimated based on similar developments or experiences. Just as a junior electrical engineer cannot architect the hardware for an IBM 360 computer, neither can a junior programmer architect or design a major software system. Breaking a large software system into small modules which can be understood and estimated requires a high degree of software talent. It is the type of talent found in experienced system analysts or senior programmers. It requires a significant design effort and cannot be done quickly on the back of an envelope.

When breaking the software package down, the estimator tries to get down to a level which is reasonably comprehensible. For example, to say that applications software will be 100,000 instructions in size is probably a guess and not an engineered estimate. However, to divide the applications software into programs and subprograms so that one can say that a certain message processing module will require 600 instructions indicates that a certain amount of engineering design and definition has preceded the estimate.

In order to break the software into small modules, the software architect or designer must understand the functions and subfunctions which are to be performed by the system software. For example, he must recognize that a certain message processing function must be performed by a software module. This assumes that the operational requirements of the user are well known and that a system specification has been prepared which identifies software and hardware functions. However, the degree to which such design detail is available to the estimator depends upon the stage of weapon system development. During the conceptual phase, little design information is available and an estimator must resort to gross estimates such as 100,000 instructions for applications software. As the system progresses through development, the amount of design information increases to where an estimator can now predict that a message processing module is required.

It is important to note that the estimator requires both talent and design information. Recognizing that a message processing module is required doesn't help unless the estimator can utilize his knowledge or experience to estimate the size of the module. Likewise, being able to size the message processing module does not mean that an estimator has either the capability or the necessary design detail to break a 100,000 instruction software package into small 600 instruction modules. Thus, these two factors of talent and design detail seriously impact the ability of the estimator to estimate the number of instructions.

Instructions As A Predictor Of Cost. If an estimator arrives at reasonable values for the cost per instruction factor and the number of instructions, the resulting cost estimate still might not be reliable. The problem is that the validity of using the number of instructions as a predictor of cost is questionable. First, the use of a cost per instruction factor does not specifically address other factors which influence software cost. A study by the System Development Corporation (SDC) for the Electronic Systems Division of the Air Force Systems Command identified 94 variables which influence software cost. These variables address not only the properties of the software and the weapon system, but also address external factors which can affect the software acquisition process. Using a simple cost per instruction factor ignores many other variables which can impact cost.

Second, the relationship between the number of instructions and cost has not been well defined by past research efforts. Much of the literature simply assumes that the relationship is linear and applies the same cost per instruction factor to developments which are 1,000 or 1,000,000 instructions in size. However, a few research efforts postulate exponential relationships between cost and instructions.

The SDC study (Farr and Nanus, 1964:242) developed the following relationship:

Man-months of Effort = (Constant) x (Number of Instructions)$^{1.5}$

On the other hand, Tecolote Research (Frederic, 1974:34) using probably the largest number of data points (i.e. 387) found the data best described by:

Cost (in FY73 $ K) = 0.079 (Number of Instructions)$^{0.84}$

(Again, however, the Tecolote researchers found such a low correlation between this relationship and the data, that they eventually abandoned almost all of their 387 data points and built a cost model around only five points.) Both the effort by Farr and Nanus and the effort by Tecolote utilized the number of delivered object instructions and yet the results differed significantly.

A third research effort by IBM (Malone, 1975:1-8) based its study on the number of source statements. Their findings yielded the following relationship:

Man-months = .00007 (Number of Instructions)$^{1.1386}$

Again, this result is significantly different from the other two research efforts.

Each of the three research efforts did not recommend the use of their findings to estimate software cost. This was due to the problems of gathering comparable data, to the small data samples, and to the low correlation coefficients which were obtained.

It is clear that the relationship between the number of instructions and cost is uncertain. Whether the relationship is linear, or exponential, and what values of constants should be used are currently uncertain. Not until a detailed and extensive research effort has verified the predictive relationship between the number of instructions and cost will an estimator be able to apply this estimating technique with a good degree of confidence.

Use Of Cost Per Instruction Technique. Despite the numerous problems associated with its use, the cost per instruction technique is and probably will remain the most utilized technique. Its prevalence is perhaps due to its simplicity and the appearance it provides of a scientific approach. It implies that an appropriate cost per instruction factor has been selected based on similar software developments. It also implies that a detailed software design has been performed to determine the number of instructions in the software package. Whether or not these two implications are in fact true should be strongly questioned by managers, engineers, and cost estimators who are presented with cost estimates based on this technique.

Specific Analogy

A second cost estimating technique is specific analogy where future software costs are estimated from the historical costs of a similar software development. This technique is frequently used in estimating the cost of manufactured goods. For example, if previous production costs for a television set are known, a manager can use this information to predict future production costs of a similar television set.

However, in applying this technique to software, there are two major problems. First, there is limited historical data concerning the software cost of past weapon system developments (Air Force OSR, 1973:46). Second, the nature of software is such that there is usually limited similarity between any two software developments.

Historical Data. To utilize the specific analogy technique, an estimator must first determine the software costs associated with a similar program. This is difficult for a number of reasons. Until recently, software costs were generally not separated from the hardware cost of a weapon system. Work breakdown structures which encourage detailed software cost and status reporting are still lacking (Ashe et al, 1975:Vol 2, 2-35).

A few programs have required that software cost be separately identified and recorded. Unfortunately, that data suffers from the lack of definition which has previously been discussed. There is no common agreement or standard which details what elements of cost are software related. For example, during a system test of hardware and software, programmers are required to develop test plans and to modify programs. Whether these costs are charged to system test or to software depends upon the accounting system and the work breakdown structure. Again, the cost elements and the phases of software development which are included in available cost data are not well defined.

Until a common set of definitions and a common data collection requirement is levied upon major weapon system developments, a software cost estimator using the specific analogy technique will probably have to rely on the use of only one or two historical data points gathered by personal contact or personal experience.

Software Similarity. After determining the software cost of a similar development, the estimator must then make a somewhat subjective set of judgments concerning the similarity of the new development with the old development. It is very unlikely that the new system will utilize the same hardware or be required to perform the same functions. Therefore, drawing parallels between systems is difficult. For example, cost estimators used the software cost of the Air Force Back Up Interceptor Control (BUIC) to estimate the software cost of the Air Force SAGE System (Jones, 1965:19).

While somewhat similar in mission, the two systems are quite different and significant subjective judgment was required to assess these differences and to assign a cost to them.

Other systems such as the Airborne Warning and Control System (AWACS) have few similar developments upon which to draw comparisons. Advancing technology continues to introduce new hardware and software techniques which compound the problem of finding a similar program to use for an analogy. Again, even if a similar program is identified, the problem of obtaining accurate historical cost data remains.

Use Of The Specific Analogy Technique. Despite these problems, the specific analogy technique is still in active use. While it is subjective in its application, it does take into account the real world problems and performance which actually occurred on a similar program. For example, if an estimator used the BUIC cost data to estimate the SAGE system, the estimator will automatically include those costs which were associated with funding difficulties, changing requirements or other real world problems.

The specific analogy technique forms the primary basis of a formal Army software cost estimating technique. An Army catalog provides detailed cost information for 20 data systems. An estimator can broaden his knowledge by utilizing the recorded experiences found in the catalog. Adjustments in the historical cost indications due to differences in the new system are then identified and explained (Department of Army, 1975).

While subjective in nature, the specific analogy method tends to minimize optimistic schedules and estimates since it is based on actual performance data for a similar system. With better collection and recording of historical software cost data, this technique holds much promise. A research effort is currently underway by the Rome Air Development Center of the Air Force Systems Command to develop the historical data base which is required for the use of this technique (Nelson and Sukert, 1974).

One frequent criticism of this technique is that it is not very objective or scientific. While an estimator who states that the applications software consists of 100,000 instructions might be thought to have a scientific basis, an estimator who states that a system will require 50% more for software is felt to be subjective and unscientific. In fact, neither technique by itself has any inherent scientific credibility or objectivity. Instead, it is the conscientious and documented application of these techniques which can make either one a reasonable tool for producing a reliable cost estimate.

Again, the lack of historical data limits the specific analogy technique. Current and future data collection efforts may eliminate this problem. The remaining difficulty will be in the subjective extrapolation of the cost of one program to another "similar" program.

Expert Opinion

A third software technique for estimating software cost is the use of expert opinion. The title is self explanatory. To estimate software cost one simply asks an expert or a group of experts to use their knowledge and experience in predicting the cost of software. Two methods for obtaining an expert opinion are the engineering cost analysis and the Delphi method.

Engineering Cost Analysis. Despite its somewhat authoritarian title, an engineering cost analysis is simply an expert opinion. A software expert or software engineer is presented with a functional description of the weapon system. He then proceeds to analyze the software usually by breaking the software into smaller programs and subprograms. When the software has been divided into small and comprehensible modules, the expert then estimates the amount of resources required for each module using his knowledge and experience as a guide. For example, the expert might identify a message processing module which in his opinion will require three months of effort by a junior programmer and six hours of central processor time. A cost analyst then transforms the estimated man-months and computer time into a dollar cost.

Some problems with this technique have previously been discussed. The technique requires an "expert" which is a vague term referring to someone who can predict software resources accurately based upon his education and experience. If experts were identified based on demonstrated estimating accuracy, the technique might be ideal. However, such identification is not common in practice and true "experts" are difficult to find.

Also, an engineering cost analysis requires that the functional design of the weapon system be fairly complete. Decisions concerning which functions are to be performed by hardware or by software have to be made before an engineering cost analysis of the software can be made. Explicit and detailed documentation of the user's requirements must be available.

Use Of Engineering Cost Analysis. Despite these problems, the engineering cost analysis has some unique strengths. It requires that an engineer sit down and design the software into small enough modules so that he can understand the resources required to develop the module. If the functional modules are fairly small (e.g. three man-months), then the program manager can be reasonably certain that a fairly detailed design has been made and that the user's requirements were known to sufficient certainty to support the detailed design. On the other hand, if the estimate makes frequent references to large modules (e.g. five man-years), then the program manager might infer that either a detailed design has not been performed or that the user's requirements were uncertain and forced the engineer to group requirements into large modules.

Thus, the use of an engineering cost analysis can provide the program manager with an insight into the degree with which user requirements are known. Since uncertain user requirements are a major cause of system overruns, the use of an engineering cost estimate may provide the manager with valuable information.

Another benefit from the use of an engineering cost analysis is that it takes into account the unique nature of the new program. The impact of certain interfaces or timing requirements can be individually addressed instead of basing their costs on the average historical costs of previous systems. For this reason and for others, the use of an engineering cost analysis is recommended by most of the researchers into the area of software cost estimating.

The Delphi Method. A second method for obtaining a cost estimate from a group of experts is the Delphi method. Developed by the RAND Corporation, Delphi first gathers estimates from individual experts. The results of the individual estimates are then iteratively fed back to the experts until a consensus is reached. The Delphi method is not a committee. The experts do not meet face to face. Instead, the experts are given the results of each iteration and asked to explain or justify their opinions. These opinions and reasons are then given to the other experts until eventually the estimates of the experts converge.

RAND performed an experiment using Delphi to estimate a particular project's software cost. Two teams of experts were used and guided by the

Delphi method. Their estimates were 217 man-months and 1090 man-months respectively. The reasons for the wide difference were not clear. Two additional groups of experts were also asked to estimate the same effort using a simple committee technique. Their results were 485 man-months and 656 man-months respectively. The actual cost was 489 man-months which would tend to indicate that the use of the committee technique might be better than the Delphi method (Farquhar, 1970).

Delphi Versus Committee. The purpose of presenting the Delphi method is not to approve or disapprove its use. The purpose is to examine why a face to face committee of experts appears to perform better than a faceless group of experts. In a study by another RAND researcher, the Delphi technique was found to be an unreliable method for estimating. He found considerable evidence that there was no difference between the estimates of laymen and experts and suggested that Delphi leads to a manipulated group suggestion and not a true consensus (Sackman, 1975). Perhaps the reason that the committee outperformed the Delphi method was that it encouraged face to face confrontation which enabled the group to judge whether someone was a true "expert" and to reach a real consensus.

The point is that when obtaining estimates from a group of experts, it appears advisable to have the experts engage face to face. Iterative formal communication between different agencies might result with estimates similar to the Delphi method. On the other hand, the use of a committee to examine and explore the differences in cost estimates might result in improved communications and better estimates.

Cost To Cost Estimating Technique

A fourth technique for estimating software cost uses the cost of one part of the weapon system to estimate the cost of another part of the system. This technique is frequently used to estimate the cost of system test or of initial spares. Both of these costs can be estimated based upon an historical percentage of the prime mission equipment. For example, initial spares for similar ground electronic systems might be an average 20% of prime mission equipment cost. Once a detailed estimate of the prime mission equipment for the new system has been made, the estimator can then simply apply the 20% factor to that cost to estimate the cost of initial spares. This technique works quite well with a limited number of items (Jones, 1965:26).

However, applying this technique to software is difficult. First, there again is the problem of insufficient historical cost information. Second, software constitutes almost 90% of some systems such as WWMCCS and only 2% of the B-1 (Ashe et al, 1975:2-46). Using an average cost factor for software simply does not work as well as it does for spares or other system components. Despite these problems there are ways in which the cost to cost technique can be used in a limited fashion.

For example, designing, coding, and testing a program might be expected to average 40%, 20% and 40% respectively of total computer program development costs. Once the design of a particular program has been completed, a manager can estimate the expected costs for the remaining activities of coding and testing. As a rough rule of thumb, this technique has some merit. However, there are significant differences in the percentages of total cost attributed to any one activity (Boehm, 1975:7). Whether these differences occur because of the lack of distinctness in the definition of the activities or whether these differences are attributable to some other cause is uncertain. Again, limited cost data and poor definition limit the accuracy and utility of this technique.

Non-Cost To Cost Relationship Technique

The fifth category of software cost estimating technique is the use of a non-cost parameter to estimate cost. Frequently called parametric estimating, it is by far the most sophisticated and ambitious of all the techniques. The technique develops parametric equations where characteristics or parameters of a new computer program are used to estimate its cost. These characteristics include such parameters as the number of source instructions, the type of weapon system, the age of the computer upon which the program will run, the type of compiler, and numerous other characteristics. One parametric research effort by the System Development Corporation (SDC) identified over 90 parameters which influence cost (Nelson, 1967).

Once these parameters are identified, the cost estimating researcher attempts to collect a large sample of data so that the relationship between these non-cost parameters and software cost can be established using regression techniques. In order to have a statistically valid parametric equation, researchers attempt to obtain as many data points as possible. This search for many data is the source of one of the problems with this technique.

For example, a researcher might have data on five JOVIAL developments, five Fortran developments and five Cobol developments. Rather than develop three models based on only five data points each, a researcher is likely to lump all of the fifteen points together and develop a single model for use on programs written in either JOVIAL, Fortran, or Cobol. While the use of fifteen points appears to give the model a greater statistical basis, an implicit assumption is that there is no cost difference due to the use of different computer languages. Further lumping of data as to the type of computer, the particular compiler and other factors is made until the parameters used in the equations are quite broad.

This lumping of data may be regrettable, however, it is the only possible course for researchers who are faced with a scarcity of data. In fact, the lumping of data has been highly successful in other areas. In estimating the production cost of an airframe, knowledge of only three parameters (e.g. weight, speed, and production quantity) are sufficient to reasonably predict the production cost of an aircraft (Levenson et al, 1971).

Software cost, however, appears to be dependent on more variables than other products. A technique using only two or three parameters suffers in predictive power. However, a technique which requires the determination of many factors is unlikely to be popular in use, especially if the determination of a proper value for the various factors is difficult. Thus, researchers not only have a problem in finding appropriate parametric equations, they also must attempt to minimize the number of inputs required for the use of the equation.

Researchers have frequently developed relationships among parameters which are quite different from the efforts of other researchers. This is perhaps due to the lack of clear definitions in the area and to the small data samples available to the researchers. However, the impact is that no single parametric technique has received a large degree of acceptance or a wide degree of use. To understand the problems with developing or utilizing parametric techniques we can examine five of them. Three of them are well publicized and in limited use. These three techniques are those by the System Development Corporation, by J.D. Aron, and by Ray Wolverton. A fourth technique is a recently developed technique used by the Computer Systems Command of the U.S. Army while the last one is one developed by

Tecolote Research for the Navy.

SDC Technique. The SDC technique was developed under contract to ESD from 1964 to 1968. The results of this massive research effort are contained in nine volumes identified in the bibliography. The Handbook written by E.A. Nelson in 1967 summarized most of the research findings.

To gather data, the SDC researchers resorted to the use of questionnaires. While this limited the accuracy of the data, they did amass 169 data points which remains the largest software cost data base currently published by any single group of researchers (Morin, 1974). The SDC researchers identified over 90 variables and examined their influence upon cost during different phases of the software development process.

For the design, code, and test phase of software development, the researchers were able to develop a parametric equation based on fourteen different parameters. Again, while the researchers examined over 90 variables, their final equation is based only on 14. This is because the remaining variables do not improve the predictive power of the parametric equation. The equation is worthwhile to examine in depth. It is:

$$
\begin{aligned}
Y(1) = &- 33.63 + 9.15\,X(3) + 10.73\,X(8) + .51\,X(26) + .46\,X(30) \\
&+ .40\,X(41) + 7.28\,X(42) - 21.45\,X(48.1) + 13.53\,X(48.5) \\
&+ 12.35\,X(51) + 58.82\,X(53) + 30.61\,X(56) + 29.55\,X(72) \\
&+ .54\,X(75) - 25.2\,X(76)
\end{aligned}
$$

Where the variables have the following interpretations:

Y(1)	- Total man-months required
X(3)	- Lack of knowledge of operational requirements
X(8)	- Stability of design
X(26)	- Percent mathematical instructions
X(30)	- Percent information and storage

X(41)	- Number of subprograms
X(48.1)	- Business
X(48.5)	- Stand alone
X(51)	- First program on computer
X(53)	- ADP components developed concurrently
X(56)	- Random access device used
X(72)	- Different computers for programming and operation
X(75)	- Number of man trips
X(76)	- Program data point developed by military organization

The purpose of reproducing this equation here is not to recommend its use. Instead, one should note the conspicuous absence of one variable from the parametric equation. The SDC equation for predicting man-months does not utilize the number of computer instructions as an independent variable. None of the other variables have a very direct relationship with the number of instructions. The lack of this variable implies that the SDC researchers found little improvement, if any, in their parametric equation when adding the number of instructions to the input. This is important to note since most other parametric estimates depend heavily (if not exclusively) on the number of instructions as an input (Morin, 1974).

Recognizing the importance of the number of instructions in other research efforts, the SDC handbook provides tables which indicate the number of man-months of effort required to design, code, and test 1000 instructions of a particular language. If we look at some entries in those tables, we might better understand the conspicuous absence of the number of instructions from the SDC parametric equation. Two of the entries are:

JOVIAL (1000 object instructions)

No. of data points*	Man-months of effort required				
	Max	Min	Std Dev	Median	Mean
15	7.6	.66	2.31	2.5	3.07

JOVIAL (1000 source instructions)

No. of data points*	Man-months of effort required				
	Max	Min	Std Dev	Median	Mean
15	46.25	2.13	12.01	6.15	10.27

*(Note: Each data point represented a separate software development project).

Thus, the man-months of effort required to design, code, and test 1000 JOVIAL source instructions was only 2.13 man-months on one software project, while it took 46.25 man-months to design, code, and test 1000 JOVIAL instructions on another program. The large standard deviation indicates that there was considerable variability in the data and it is easy to understand why the SDC researchers did not inculde the number of instructions as a parameter in their equation.

Unfortunately, by publishing a mean or expected number of man-months for 1000 JOVIAL instructions, the researchers provided information which a novice cost estimator might take out of context. The variance in the data indicates that the number of instructions is either a poor estimator of man-months or that the SDC data was faulty. Under either circumstance, use of the mean from this table would be of questionable value. This is pointed out because in Chapter 5 we shall see that some estimators do use this mean.

While the SDC data may be questioned, it represents a major research effort in an area where few true research efforts have been made. It still represents probably the largest software cost data base ever assembled by a research team. The large number of variables examined as well as the researchers' comments concerning the impact of these variables upon cost is interesting and valuable. Unfortunately, the SDC research was not continued until a large enough data base could be developed from controlled data to produce parametric equations with reasonable confidence intervals (Nelson, 1967).

Aron's Technique. Published in 1970, Aron's technique is frequently cited in the literature. His parametric methodology not only identifies the variables, but also includes recommended values for these variables based on a large number of major IBM system programs. While Aron does not publish the data upon which his technique is based, his reference to a large number and variety of IBM programs adds a certain credibility to his article.

The Aron technique begins with a system design and an estimate of the number of deliverable, assembly level instructions. The difficulty of these instructions are then judged to be easy, medium, or hard primarily based upon the number of interactions a particular program will have. Next, the estimator utilizes the data reproduced in Table 1. The table indicates, for example, that for a hard program being performed in 12-24 months, one can expect a productivity of 125 assembly level instructions per man-month. After applying the appropriate productivity factors to each program, the result is multiplied by a factor of 2.5 to account for management and support personnel.

	Duration / Difficulty	6-12 Months	12-24 Months	More Than 24 Months	
Row 1	Easy	20	500	10,000	Very Few Interactions
Row 2	Medium	10	250	5,000	Some Interactions
Row 3	Hard	5	125	1,500	Many Interactions
	Units	Instructions per Man-Day	Instructions per Man-Month	Instructions per Man-Year	

Table 1

Productivity Table

This is an extremely brief simplification of Aron's 12 page article. The primary point is that Aron's technique is heavily dependent on the number of instructions. Also, two things can be noted from Table 1. First, there is a large range of programmer productivity based on a somewhat subjective evaluation of the degree of difficulty associated with a program. Second, Aron states that if a project extends over two years, then programmer productivity increases due to a learning effect. Aron offers no proof of a learning effect and no other available literature supports this. However, taking Aron's data at face value, one can see that programmer productivity goes from a low of 20 instructions per man-day for an easy program on a short schedule to a high of 33 instructions per man-day (10,000/year) for an easy program on a longer schedule (Aron, 1970).

Wolverton Technique. A third technique was published by Ray Wolverton in 1974 based on experience at TRW. The 21 page article describes the technique in some detail but no specific mention is made of the data base used in developing the technique. Also, no indication is given as to how well the estimating technique fits the data base.

The technique begins with a design in which software modules are identified and then categorized. The categories are old and new for the first split and then easy, medium, and difficult for another split. This yields six categories of difficulty. The modules are then further separated according to the type of function being performed. Six functions such as control and input/output are identified. This further divides the software into 36 different categories based on newness, difficulty and type of program.

Wolverton then provides a table of cost per instruction factors for each of the 36 categories. The number of object instructions in each of the 36 categories is then multiplied by the appropriate cost per instruction factor to yield a total cost including overhead. The cost per instruction factors range from $15 to $75 depending on the category.

A primary strength of the Wolverton technique is its simplicity. It requires some fairly simple subjective judgments to place the software into 36 categories. Similar to the Aron technique, it only uses a few characteristics of the software which are reasonably well known with the exception of the number of instructions. In fact, the major problems with the technique are that it requires an estimate of the number of object instructions and that it assumes a linear relationship between cost and the number of instructions within each of the 36 categories.

While the technique is attractive because of its simplicity, its use is questionable since no indication is provided as to the nature of the original data. How well the technique explains the original data should be known before one can use a technique with any confidence (Wolverton, 1974).

<u>ADPREP Technique</u>. A fourth technique has been developed for the U.S. Army Computer Systems Command by the Planning Research Corporation (PRC). The technique is called Automatic Data Processing Resource Estimating Procedures (ADPREP). The technique is based on data collected from 20 Army data automation systems. Half of these systems were new while the remainder were major revisions of existing systems. The systems were business type systems and required a level of effort averaging 105 man-months.

The technique provides a number of estimating guidelines. For example, detailed estimating worksheets addressing each phase of development are provided. An estimator can review the historical values of the 20 programs and determine an appropriate worksheet value based on analogy or judgment. Numerous heuristics for certain specific activities such as documentation are also provided.

However, an essential part of ADPREP are the parametric equations which were developed using the data from the 20 Army systems. We will review one of those equations which addresses the same variable as the SDC equation. The particular equation was developed based on the data from 10 new Army systems and predicts the total man-months of effort to design, code, and test the software. The equation is:

$$Y(2) = 2.57\,X(2) + 5.10\,X(3) + 0.12\,X(5)$$

Where the variables have the following interpretation:

Y(2) - Personnel requirements in man-months

X(2) - Total number of different output formats of ADPS products

X(3) - Total number of record types in data base

X(5) - Average number of transactions per month of input in thousands

Note that in contrast to the 14 variables used by the SDC equation, the ADPREP requires only three variables to predict man-months. Also, like the SDC equation, the ADPREP equation does not rely on the number of instructions in developing its estimate. Again, while some techniques are strongly based on the number of computer instructions, other parametric techniques do not utilize it at all.

The degree to which the estimating equation explains the sample data is also provided in the ADPREP documentation. The ADPREP manual claims that the above estimating equation has a squared multiple correlation coefficient (R^2) of 1.0. Such a high correlation coefficient makes the ADPREP technique very suspect. In essence, the ADPREP claim is that the estimating equation explains all of the variance found in the sample data and that all ten random data points were exactly on the line defined by the estimating equation. An exact fit is quite questionable considering the random nature of the variables concerned.

Nevertheless, the ADPREP technique is quite well presented and is simple to use for an estimator. The ADPREP manual warns the estimator against using the estimating equations outside of the range of the sample data. The worksheets insure that a detailed look is made at every aspect of the software development. However, the ADPREP technique also assumes quite a lot of detailed knowledge is available to the estimator. For example, the estimator must know the number of record types in the data base. This is reasonable for the business type of Army systems for which the technique was developed. However, for a real time weapon system, such variables are normally unknown for a long time (Dept of Army, 1975).

Tecolote Technique. A fifth parametric technique was developed by Tecolote Research for the Office of Naval Research. At first, the researchers gathered a data base of 387 data points. The data included the 169 data points from the SDC study as well as data from TRW, NAA Autonetics and 12 other sources. After gathering this mass of data, the researchers found that it was impossible to interpret the data because much of it was from older sources with no available spokesmen to interpret the meaning of the various data elements. This problem is common in an area where lack of definition of terms such as cost and instruction make most of the available data essentially useless for research.

The Tecolote researchers then abandoned the massive data base and selected only five points about which they had reasonable information. They then proceeded to regress the number of man-months of effort against a number of different variables. They proposed their provisional technique as an engineering scaling law rather than strictly derived statistical equations.

The Tecolote effort produced a number of estimating equations. One set of the equations predicted the number of direct labor man-months necessary to totally develop a software package - from defining users requirements to validating the software. The set of equations utilizes different input variables to output man-months. For example, if one knows the number of air threats a weapon system will be required to track, the following formula is provided:

$$\text{Total man-months labor} = 69\ (\text{No. of targets})^{1.88}$$

In this case, the estimating parameter is a functional characteristic of the weapon system and not of the software.

On the other hand, if one knows (or can estimate) the number of operating instructions, the following equation is provided:

$$\text{Total man-months labor} = 2.52\ (\text{Thousands of instructions})^{1.24}$$

This equation relates a software parameter to software cost.

The notable thing about the Tecolote effort was their effort to relate both weapon system parameters and software parameters to cost. While the reliability of the data is admittedly questionable, the resulting equations provide a manager with an indication of how cost might vary with operational weapon system parameters. With this type of information, a manager might seek to lower software cost by trading off the operational requirements of the system (Frederic, 1974).

Use Of Non-Cost To Cost Technique. Non-cost to cost techniques are admirable in their attempt to consider the many variables which can influence software cost. However, as can be seen from the five parametric techniques that have been described, research into the various cost estimating relationships is far from conclusive. While some techniques rely almost totally upon the number of instructions, some researchers have found little use of the number of instructions in predicting cost. While some researchers have postulated linear relationships between some variables, others have postulated different exponential relationships. Until more research has been made with controlled data, the current parametric techniques provide their user with little confidence.

Perhaps the problem with the various techniques can be better understood by examining the various programmer productivity rates which different researchers have found. Aron, for example, indicated that a programmer can produce from 125 to 500 object instructions per month based on numerous IBM projects. Wolverton's price per instruction matrix indicates a productivity range of 62 instructions to 312 instructions with a center value of 156 instructions per month. The SDC researchers found that the productivity for object instructions ranged from 10 instructions per man-month to a high of 7142 instructions per man-month on different projects. With data which indicates such wide differences in productivity rates, it is easy to see why researchers have had problems in reaching similar conclusions.

Again, whether the differences are due to the lack of controlled and comparable data or whether the differences can be attributed to numerous other causes is still uncertain. Until a major data collection effort is made of well defined data, the wide spectrum in the findings of parametric researchers leaves many questions unanswered.

A State Of The Art Assessment

We have examined five different categories of software cost estimating techniques. Each of these techniques has certain problems in its use and its accuracy. These problems are reflected in the software developments of both industry and government. Underestimations are extremely common in most developments while overestimations are unheard of. While a number of efforts have been made to develop improved cost estimation techniques, no generally reliable software cost estimating technique exists.

This chapter has reviewed only a few of the major software cost estimating efforts. For a more detailed explanation of many research efforts and techniques, the reader is referred to a master's thesis written by Lois Morin under the advisorship of Frederic Brooks at the University of North Carolina. The thesis is well written and quite comprehensive it its assessment of various published software cost estimating techniques.

Decisions And The State Of The Art

The problems with managing software are quite well appreciated. Software development has been a critical problem in most weapon system developments and is currently a major topic of interest in the systems acquisition business. Having seen the types of decisions which rely heavily upon a software cost estimate and having seen the state of the art of software cost estimating, perhaps one of the causes of the software management problem can be better understood and appreciated. Why we select the wrong system, the wrong computer, or the wrong contractor can ofter be traced back to a decision maker's inability to reliably estimate software cost. Why we fail in the development of major systems such as the Advanced Logistics System or why we are surprised by huge software overruns is also perhaps partially explained by the inability to predict software cost. Again, if we seek to improve the quality of our decisions concerning weapon system software, then we must be able to predict software cost.

Before a manager can hope to control the cost of software, he first must understand how software costs are generated. He must understand the relationship between weapon system parameters and software cost. He must be aware of how and whether the number of instructions is related to cost. Until he understands these relationships, his control of software cost will be haphazard and unguided.

IV. Methodology

The purpose of this research is to provide managers, researchers, and cost estimators with insight into the software cost estimating process. To accomplish this, the software cost estimating environment at the Electronic Systems Division (ESD) was examined to determine what techniques were being used, how they were being applied, and what problems were being encountered. This chapter discusses the methods used to gather and analyze information concerning the environment at ESD.

Working Hypotheses

In essence, this research is descriptive in nature. We need to know what the current environment is in order to live in it, to understand it, and to improve it. To describe every feature of the software cost estimating environment at ESD was beyond the scope of this ten week effort. Therefore, working hypotheses were formed to focus on the more interesting and important facets of that environment. These working hypotheses served to guide the research and limit the area of investigation.

Again, the hypotheses were only used to guide and limit the research. They are not hypotheses in the formal, statistical sense. No attempt was made to statistically accept or reject an hypotheses. Instead, a guess was made concerning the environment and evidence was gathered concerning that guess.

The hypotheses are stated here in "null" form. That is, the writer expected to collect data which would tend to refute each of the hypotheses. The hypotheses are:

a. There is a common method used in estimating the cost of software.

b. The persons responsible for making software cost estimates have similar backgrounds.

c. The software cost estimate is made when most of the software cost drivers are known.

d. A management reserve for software is maintained.

e. The independent cost estimate provided by an outside agency is truly independent.

f. Software estimates are challenged.

g. The confidence intervals which software cost estimators place on their estimates are narrow and uniform.

h. There is uniformity in the elements of software cost addressed by a software cost estimate.

i. The contractor's cost proposal indicates the method used to predict the cost of software.

j. Contractors' estimates in response to a software Request For Proposal (RFP) do not vary widely.

k. The type of software cost data currently being collected on software contracts will support the types of cost estimating methods currently being used.

l. A primary management indicator of software status is the comparison of budgeted cost and actual cost. The percent complete of the software task is equal to the ratio of actual cost to budgeted cost.

m. The software estimate does not change.

A limited pretest of the interview was made at Wright-Patterson AFB. The pretest identified some questions that were vague and required refinement. A revised set of questions was prepared for use in the ESD interviews. The actual interview instrument is contained in Appendix B.

The interview contains a large number of questions. The reason for this was twofold. First, the research effort was attempting to describe an environment. To adequately do that, as many facets of that environment which could reasonably be described had to be explored. Second, the software acquisitions at ESD are at different stages. Some programs are not yet under contract and would not be able to answer questions concerning contractor performance. It was expected that few of the interviews would be able to address every area. In order to insure that an adequate amount of information was obtained, a large number of questions were developed.

The Interview Subjects

With the interview instrument complete, it was then necessary to identify the interview subjects. There are 21 programs at ESD which involve the acquisition of a major software package. With the assistance of Major Richard Grimm and Captain Gerald Bourdon of the Cost Analysis Division, a letter (Appendix A) was sent to each program office asking them to establish a point of contact for the interview. The letter requested that the point of contact be the person most knowledgeable of the software cost estimate made for each program.

Administering The Interview

The interviews were personally administered by the writer during the period 12-23 April 1976. An attempt was made to interview each of the 21 points of contact during that period. However, the limited time available, coupled with the non-availability of some personnel, resulted in only 12 interviews.

In some cases it was possible to gain some limited information concerning a program without conducting an interview. This was done primarily by reviewing contractor cost performance reports (CPR) for those programs on contract. Therefore, with the interviews and the cost performance reports, some data was gathered on 16 of the 21 programs.

Table 2 indicates the programs at ESD which involve a major software acquisition. The nature of the data collected is also indicated. Note that the interviews were conducted in almost all of the ESD deputates and were not limited to any one mission area.

Administering each interview took from 30 minutes to three hours. The wide span of time resulted from the different nature of the programs. In every case, the interview subject was totally cooperative.

Notes were taken at the time of the interview on the responses to the four open ended questions. These notes were expanded at the end of each day.

Interview Responses

As expected, few of the interview subjects were able to address each of the question sets. In most cases, this was due to the nature of the program. Again, subjects whose program was not yet on contract could not respond to the questions addressing contractor performance. In other cases, the interview subject was unfamiliar with certain aspects of the program and unable to readily obtain the necessary information. The net result is that while some of the question sets were answered by as many as 13 subjects, some of the question sets were answered by only three.

In the chapter on research findings, some reasons are offered to explain the limited response for a particular question set. It is hoped that these reasons will indicate areas in which information is sparse and serve to guide future researchers.

	PROGRAM OFFICE AND OFFICE SYMBOL	INTERVIEW	COST PERFORMANCE REPORT
1.	Airborne Warning and Control System (AWACS) - YW	YES	YES
2.	Advanced Airborne Command Post (AABNCP) - YS	YES	N/A*
3.	World Wide Military Command and Control System (WWMCCS) - WW	NO	NO
4.	SAC Automated Total Information Network (SATIN IV) - MCV	NO	N/A*
5.	NORAD Cheyenne Mountain Improvements (427M) - MCN	YES	YES
6.	Air Force Enlistment and Entrance System (AFEES) - MCH	YES	YES
7.	TRACALS - OCN	NO	NO
8.	Combat Grande - OCW	NO	YES
9.	Cobra Dane - OCD	YES	NO
10.	Over The Horizon Radar (414L) - OCS	NO	YES
11.	Joint Surveillance System (JSS) - OCU	PARTIAL	N/A*
12.	Pave Paws - OCL	YES	N/A*
13.	Tactical LORAN - DCL	YES	YES
14.	Satellite Communications (AFSATCOM) - DCK	YES	N/A*
15.	TACS Improvement (485L) - DCY	YES	YES
16.	TIPI Image Interpretation (TIPI II) - DCM	YES	YES
17.	TIPI Tactical Electronic Processing and Evaluation (TIPI TERPE) - DCM	YES	N/A*
18.	TIPI Display Control/Storage Retrieval (TIPI DC/SR) - DCM	NO	YES
19.	Joint Tactical Information Dissemination System (JTIDS) - DCB	NO	NO
20.	Combat Theater Communications - DCJ	NO	NO
21.	Office for the Application of Special Intelligence Systems (OASIS) - XRI	YES	N/A*

*Indicates that the Cost Performance Report was not available either because the effort is not yet on contract or that cost reporting on software was not a contract requirement.

Table 2

Sources of Data

Analysis

With the data collection complete, the next ta˴ was to examine or analyze the data to determine if it tended to support or refute the initial hypotheses. Again, the nature of this research effort is descriptive. No causal relationships were hypothesized or explored.

The technique used to examine the data and the hypotheses is quite simple. In the following chapter on findings, each hypothesis and its related question set are examined individually. The hypothesis is stated and then discussed to insure a clear understanding of the hypothesis and its impact.

After the hypothesis is understood, the question set used to collect data is examined, and the data sought by each of the questions is explained. Since the sources of data varied for each question set, the programs for which data was available are identified. Questions for which limited or no response was obtained are re-examined to explain why the response was limited. This may aid future researchers and also serves to indicate areas in which information is sparse.

With the hypothesis and the question set understood, the responses are then examined. In some cases it is only necessary to look at some aggregate characteristics of the data such as the range and the median. In other cases, it is desirable to examine each of the individual responses. The technique used for examining each set of responses is adapted to the peculiar nature of the questions and the types of responses.

Finally, with both the hypothesis and the data understood, a determination is made whether the data tends to support or refute the hypothesis. Admittedly this determination is somewhat subjective. However, the reader is provided with sufficient information concerning the hypothesis and the data so that he can reach his own conclusions concerning the validity of the hypothesis.

Again, each hypothesis and question set are examined individually. Both failures and successes in collecting data are explained to assist future researchers.

No attempt was made to correlate the data from one question set with the responses from another question set. Since most question sets were answered for different subsets of programs, any such correlation would have been misleading.

V. Research Findings

This chapter presents the results of the interviews conducted at ESD. The data collected against each of the thirteen hypotheses is examined and analyzed. Each of the hypotheses forms the basis of a research finding. Each finding is addressed individually, using the following format.

Hypothesis

For each of the research findings, the initial working hypothesis is reviewed and discussed. The impact of accepting or rejecting the hypothesis is examined in broad detail. Again, each hypothesis is stated in null form. That is, the writer expected the data to refute the hypothesis.

Sources Of Data

Since not all of the subjects interviewed were able to address each hypothesis, the sources of data for each of the findings is specified. This should give the reader a better feeling for the validity or relevance of a particular research finding.

Questions And Responses

For each of the findings, the interview questions which were initially prepared are examined. Problems which were encountered with certain questions are discussed. Hopefully, this discussion may aid future researchers into the area.

The responses for each of the questions are then presented. The range of responses, the mean response, and other appropriate characteristics of the data are also presented.

Discussion

After the data for each finding is presented, a discussion examines the impact the data has upon the initial hypothesis. Again, no formal or statistical test of the hypothesis is performed. Instead, the data is again examined to see if it tends to support or refute the hypothesis.

Finding #1 - Software Cost Estimating Methods

Hypothesis

Chapter III examined a wide variety of techniques which can be used to estimate the cost of software development. A research hypothesis was formed to examine what techniques were being used and how they were being applied at ESD. The hypothesis was, "There is a common method used in estimating the cost of software."

Sources Of Data

Data were gathered from thirteen programs representing a wide cross section of ESD programs. Due to the sensitive nature of some of the responses, the individual programs are not identified.

Questions

Two open-ended questions were used.

Question #1. "Briefly describe the software acquisition associated with your program. What is being acquired? When? How?" Many of the programs at ESD have more than one software acquisition underway. The intent of this question was only to identify a single software acquisition to serve as the object of the following questions. In cases where a program had more than one software acquisition, the one selected for discussion was the one with which the respondant was most familiar.

Question #2. "Describe how the program office estimate of software cost was obtained." The thirteen responses to this question varied greatly in both the amount of information available and the nature of the available data. Personnel transfers frequently made contact with the original estimator impossible.

Rather then simply classifying the responses, the wide variety of information collected merits examining each of the thirteen responses. Again, due to the sensitive nature of some of the responses, the programs are not referred to by name.

Program A - The person interviewed had considerable previous work experience as a programmer and as a supervisor of programmers for a major software contractor. Using this experience, he was able to analyze the total software package and divide the software into modules. These modules ranged in estimated size from 600 instructions to 35,000 instructions. The average module, however, was about 3,000 instructions in size.

Having developed an estimate of the number of instructions, the estimator then referred to the SDC study which has previously been discussed (Nelson, 1967). He did not use the parametric equations recommended by the SDC researchers. Instead, he selected a single figure from one of the SDC tables which indicated the mean productivity of programmers in terms of JOVIAL instructions per month. This particular table, and the associated problems with the use of its mean value, were previously discussed in Chapter III. Again, the SDC data was based on only 15 projects and indicated a very wide variance in productivity experienced by different programs. Nevertheless, the estimator selected a value which indicated an average productivity of 325 JOVIAL instructions per month. This value appeared reasonable to him based on his personal experience and based on his knowledge of the particular software package. Using this figure and the estimated number of instructions, an estimated cost of software was produced.

Since this program is nearing completion, the estimator was able to comment on the accuracy of his prediction. For example, his initial estimate of the number of instructions in the applications software was 160,000. Three years after the estimate was made, the current instruction count is 156,000. The accuracy is impressive.

Not only was the estimator accurate in his estimate of size, but also in his estimate of cost. The current cost to complete is only 20% higher than the original three year old estimate. In light of inflation, this cost estimate is again impressive.

Thus, despite the questionable source of an average productivity figure, the estimator produced an accurate estimate. It should be noted, however, that the respondent had nearly 20 years of work experience in programming. This work experience might account for his accurate estimate of a multi-million dollar software effort.

<u>Program B</u> - This program called for the development of a new system to replace an existing older system. A special team was formed for the express purpose of sizing the computer hardware necessary for the new system. To size the hardware, the team first reviewed the software of the existing system and the improvements expected of the new system. By analogy, an estimate was made of the size of the software. This estimate was made only for the purpose of sizing the computer hardware. The team chief did not believe that the estimated number of instructions was a valid parameter for estimating the cost of the software. Nevertheless, the estimated number of instructions was later provided as the primary input to another cost analyst to determine the cost of software. Using parametric techniques such as the Tecolote model, the estimator predicted software cost almost solely based on the number of instructions.

Again, the team that developed the estimate of the number of instructions did so by analogy and not by an engineering analysis. Their intent was only to size the hardware. Regardless of this, once an estimate was available concerning the number of instructions, translating the number of instructions to a cost was almost inevitable.

One possible reason why the program jumped at the chance to estimate cost based on the number of instructions might be found by looking at the previous software cost estimates for this program. Prior to the estimate of the number of instructions, an estimate of software cost had somehow been derived. How the number was obtained could not be determined. However, during a nine month period of management review prior to contract award, the initial estimate increased in large steps until the estimate was three times the original estimate. Thus, when faced with the opportunity to obtain an estimate which would appear bona fide, the program office jumped at the chance. Again, estimating software cost solely on the number of instructions offers questionable accuracy. However, it does give the estimate a seemingly scientific basis and makes the estimate more readily acceptable. To challenge the new estimate, one would have to challenge the estimated size of the software - a major task. Faced with continuing challenges and changes in the software cost estimate, it was to be expected that the program office would jump at the chance to develop an estimate which, while possibly not accurate, would be difficult to challenge.

<u>Program C</u> - The software estimate for this program was prepared by a program office engineer who was well supported with engineering and design services from a support contractor. A complete design of the new system had been contracted for which identified seventeen software modules and their function. By discussing each of these modules with engineers from similar

systems, the estimator gained an appreciation of the effort involved in each module.

An estimate was then prepared for each module in terms of man-months. The estimate for each module was given as a range. For example, the expected or most likely effort required for a module might be four man-months. The least likely or worst case estimate for the same module might be six man-months. The estimate for the module was then given as 4 to 6 man-months. The modules ranged in average size from 3 to 12 man-months. The total expected value was 138 man-months with a worst case estimate of 205 man-months. An additional 25% was added to the estimate to account for undefined modules resulting in a total estimate of 173 to 256 man-months. This was then converted into a cost range of $830,000 to $1,270,000 by multiplying by a "loaded" man-month. A loaded man-month includes not only the direct labor cost of programmers, but also the overhead costs associated with support personnel and facilities.

When this estimate was presented to the independent estimator, there was total agreement with the method used to estimate the software cost. The Cost Analysis Division approved the software cost estimate. However, the same approval was not forthcoming from higher management levels. Within the program office, it was felt that the estimate did not adequately take into account the effort required to document and test the software. Therefore, the program office increased the original estimate to $2.4M, effectively doubling the worst case engineering estimate. Again, this doubling was due to the general uncertainties surrounding software cost and to an uncertainty concerning what cost elements were addressed by the original estimate.

When the program office briefed their estimate at a higher level, a recommendation was made and accepted to even further increase the software estimate to take into account program uncertainties. In effect, the most likely estimate of $830,000 was raised to $3,800,000 by two management levels who were anxious to insure that sufficient funds were budgeted for the effort.

Again, the intent here is not to critique the method used to arrive at the estimate. In actuality, any of the estimates may eventually prove to be correct. The point here is that a well thought out and prepared engineering estimate was increased by the heuristic of doubling it not once, but twice. While this may insure that sufficient funds are budgeted for this program, the high estimate may cause other decision makers to reach incorrect conclusions. For example, the $3.8M estimate may indicate to some planners that the proposed system is not cost effective. The large estimate may cause the program office to develop a large software staff from limited resources when in fact the effort may turn out to be only 25% of the estimated size. Thus, while the program's budget may be sufficient, other software management decisions may suffer from the inflated estimate.

Program D - There was no data available concerning the original cost estimate for this system. However, in the original contractor proposal, the software was estimated to be 20,000 instructions in size. Despite this small size, the computer bid by the contractor had a core capable of handling 132,000 instructions, far in excess of requirements indicated by the estimated 20,000 instructions. However, two years after contract award, the 20,000 instructions had grown to 174,000 instructions. This increase in instructions by almost 900% was now overloading the core of the computer requiring the contractor to overlay the software.

Program E - Chapter III discussed some of the problems that can arise when using the cost per instruction technique. Essentially, the use of this technique is subject to large errors due to the poor definition of the terms "cost" and "instruction" as well as to the questionable relationship between cost and the number of instructions. Program E offered some clear insight into this problem area.

To estimate the software cost of this major acquisition, the program office first obtained an estimate of the size of the software from MITRE. This estimate broke the software down into functional modules ranging in size from 200 instructions to 50,000 instructions. The large size of some of the modules reflected the limited time available for developing the estimate as well as the uncertainties associated with certain functions. To insure that the instruction count was not overly optimistic the program office multiplied the MITRE estimate by a factor of 1.5.

With an instruction count in hand, the program office then sought to identify an appropriate cost per instruction factor to use in estimating cost. With the help of a cost analyst, they decided to use the cost per instruction factor exhibited by the AWACS program.

In computing the AWACS cost per instruction factor, the lack of clear and well understood terms was evident. The AWACS operational flight program had originally been estimated to be 158,000 instructions, but had eventually grown to 311,000 instructions in size. While the operational flight program was an important component of AWACS software, it was only a small percentage of the total AWACS software acquisition. Diagnostics, test routines, compilers, and other software programs were also procured. However, when developing a cost per instruction factor, the cost analyst only considered the number of instructions in the operational flight program.

Looking at the total software cost divided by only the number of instructions in the operational flight program yielded the analyst two figures. If one looked at the originally estimated 158,000 instructions, the estimated cost per instruction was $139. On the other hand, if one considered the 311,000 instructions which were finally delivered in the operational flight program, then the cost per instruction was only $79. The analyst provided both of these figures to the program office.

The program office was unaware that the cost per instruction figures were based on only the instructions in the operational flight program. They decided to be conservative and use a figure of $150 per instruction. They then applied this factor to all of their instructions including diagnostics, executive, communications processing, and operational software. The resulting estimate was $33M, a major software effort.

In a recent MITRE report (Ashe et al, 1975), the cost per AWACS instruction is given as a range of $6 to $13.50 per instruction. However, by counting only the instructions in the operational flight program, the AWACS cost per instruction factor rose to a high of $139. This 1000% to 2000% increase is representative of what can happen because of inconsistent definition of terms. If the program office had used the published $6 figure in the MITRE report, the $33M estimate would have been reduced to $1.32M.

Again, the intent is not to criticize the estimators. The lack of common definitions in this case led to a breakdown in communications between the cost analyst and the program office. Once this problem was identified, rapid action was taken to correct the error. The problem here is that these errors may not always be discovered.

Program F - This program is currently concerned with engineering changes to a major software system. The software managers do not estimate software cost internally. Instead, they first allow the contractor to prepare a technical and cost proposal for each engineering change. They then concentrate their effort on analyzing the contractor's software estimate, instead of trying to compare it to an internally generated estimate.

Program G - In this case, the program office cost estimate was prepared jointly by the program office and the ESD Cost Analysis Division. The software was divided into modules and an estimate of the number of instructions was made. Based on a phone call to a former software instructor, the estimators selected a productivity factor of ten instructions per day per programmer. A software cost was then computed.

While the source of the productivity factor might seem strange, the factor was obtained from someone experienced in the software area. Few published reports of programmer productivity for different software developments are available. Therefore, estimators frequently have to rely on their experience or the experience of others to determine a value for certain factors. The resulting estimate was reasonably accurate. The total cost of software at completion was only 30% higher than the initial estimate.

Program H - The estimator on this program divided the software into seven modules. He then estimated the resources necessary for each module. The modules were quite large in size. For example, the smallest module required five man-years. If we assumed that a programmer could produce ten instructions per day, the smallest module was 12,000 instructions in size. The large modules may indicate a lack of detailed knowledge of the software requirements.

Program I - The estimate for this program was prepared by the Rome Air Development Center. No details on how the estimate was derived were available to the program office.

Program J - When initially planned, the program office did not anticipate the use of software in the design of the system. However, after contract award, the contractor made a design decision which required the use of software. No software estimate was prepared by the program office and no software cost data was obtained by the program office.

Program K - The estimator for this program used three different techniques to estimate software cost. First, an estimate was made of the number of instructions by MITRE and program office engineers. Then an engineering estimate of resources was made for each of the software modules. This yielded the first cost estimate.

A second cost estimate was made by simply multiplying the estimated number of instructions by a cost per instruction factor. The factor was $44 and represented the previous experience of the program office.

A third estimate was made by analogy with a similar project. In all three cases the resulting estimates were similar. The use of three different approaches provided the program office with a higher level of confidence in their estimate.

Project L - This program used two techniques to estimate cost. First, the number of instructions was estimated by the sole source contractor. The estimator then utilized the table provided by the SDC study concerning programmer productivity. Using these two factors, an estimate of the software cost was made.

A second estimate was made by analogy with a similar pilot effort. Engineers felt that the software for the current effort was 4 to 8 times larger than the pilot effort. Based on this analogy, a second estimate was prepared. Both estimates were similar.

Program M - No detail was available concerning the program office's original estimate. However, the contractor's cost proposal did provide some interesting information. The contractor divided the software into modules and into types of effort per module (e.g. design, code, document). He also estimated the number of instructions required for each module. He then applied a productivity factor of three man-hours per instruction for design, code, and debug of an instruction. An additional $15 per instruction was added to cover the cost of software documentation.

This method is interesting because the contractor's productivity factor was supported by data from a previous ESD contract. While the productivity is much lower than other published factors, it represents measured experience on a similar program.

Discussion

It was quite obvious that no two of the thirteen programs used the same approach to estimate software cost. The estimated cost per instruction for a JOVIAL source instruction ranged from a low of $6 to a high of $150. The detail in the estimated number of instructions varied from modules 200 instructions in size to 50,000 instructions in size. Productivity figures ranged from three source instructions per day to fifteen source instructions per day. Not only were different methods used, but even those programs which used similar methods used quite different values for the various factors.

The lack of a common technique may cause a problem for a decision maker. Using two different techniques, similar software efforts can be costed quite differently. The range in cost for an estimate, depending on the method selected, can be as great as 10 to 1. Therefore, unless two software efforts are estimated by the same method, the decision maker will be uncertain as to the comparability of the two estimates.

The lack of a common technique may be due to a number of reasons. First, no single technique has yet been proven to be better than any other technique. Second, there is limited visibility into the area of software cost estimating. No common data base exists where an estimator can compare his technique with those used by other programs. Some techniques are not well publicized and may be unknown to some of the estimators.

Finding #2 - Software Estimators

Hypothesis

The educational and work background of an individual determines to some extent the types of software cost estimating techniques he can readily utilize. For example, someone who has never worked as a programmer would find it difficult to judge whether programming a particular software module would be easy or hard. Also, someone with only a basic course in Fortran would find it difficult to architect or design a major software system of 100,000 instructions.

An hypothesis was formed to examine the backgrounds of software cost estimators at ESD. The hypothesis was, "The persons responsible for software cost estimates have similar backgrounds." The hypothesis was formulated to determine whether a common level of educational and work experience existed among ESD software cost estimators. If researchers were aware of the knowledge and experience limitations of the ESD software cost estimators, they might better be able to develop a technique which could more readily be used.

Sources Of Data

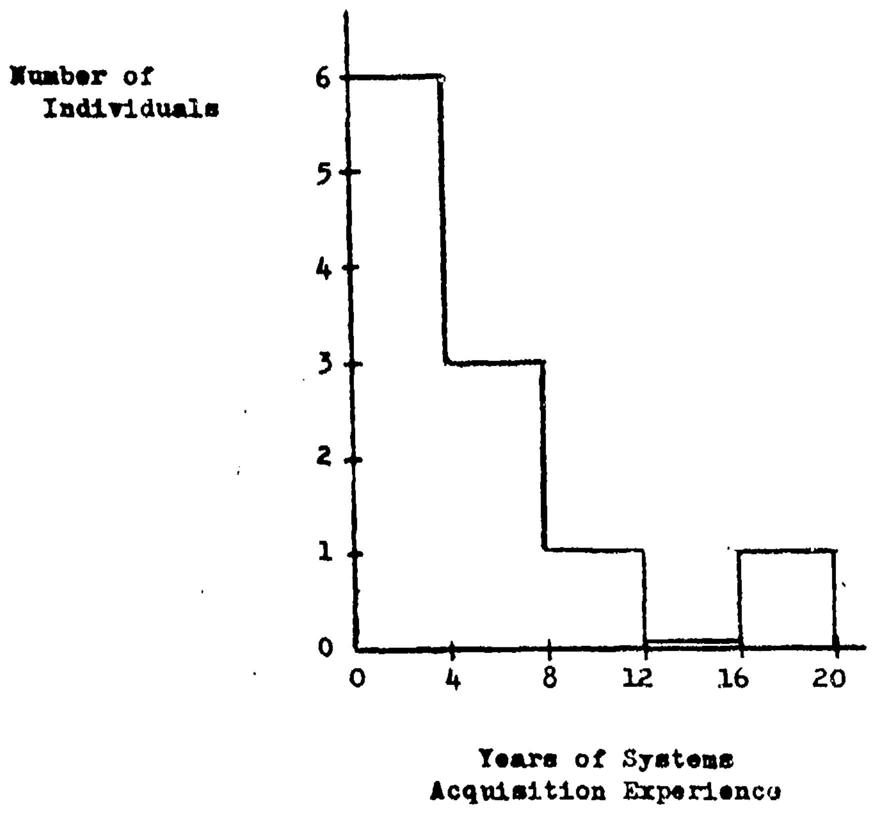
Data was collected from eleven individuals representing the following program offices: 485L, TERPE, 427M, TIPI II, LORAN, AWACS, AFEES, OASIS, AABNCP, Pave Paws, and Cobra Dane. It should be remembered that the data was collected from the person most knowledgeable of the software estimate prepared for his program. Due to personnel transfers, this person may not have prepared the actual cost estimate for his program.

Questions And Responses

Eight questions were asked in this area. Eleven persons were able to respond to all of the questions.

Question #3. "Briefly describe your educational and work background." The question revealed that all eleven individuals had college degrees in the scientific or engineering disciplines. Eight of the eleven had masters degrees. There were eight military personnel ranging in rank from first lieutenant to major. The three civilians were GS-12/13 level.

Question #4. "How many years have you been involved in systems acquisition?" Figure 1 displays the responses to this question.



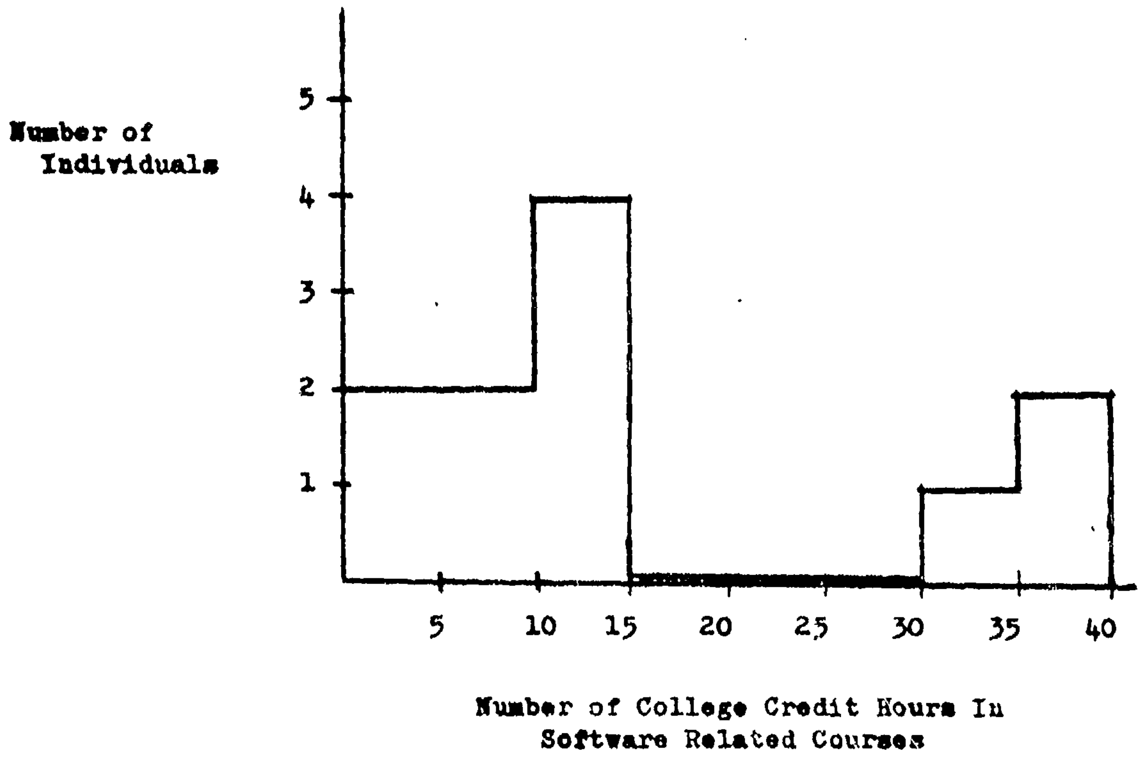
Systems Acquisition Experience

Figure 1

It can be seen that more than half of the individuals had less than four years of systems acquisition experience, while nine of the eleven had less than eight years.

Question #5. "How many years have you been involved in the acquisition of computer software?" The responses to this question were essentially identical to those of the previous question. Apparently the individuals systems acquisition experience is identical to their software acquisition experience.

Question #6. "How many college credit hours of software related courses have you attended?" Figure 2 displays the responses.



Formal Software Education

Figure 2

The range was quite large. Two individuals had no software courses while three had more than thirty credit hours.

Question #7. "Have you ever worked as a programmer? How long?" Note that this question sought to identify the individuals whose full time job was programming. While many of the respondents could write a computer program, only four had ever worked full time as a programmer. The range in their programming experience was from one to five years.

Question #8. "Have you ever directly supervised a group of programmers? How long?" The same four individuals who had been programmers had also worked as supervisors of programmers. Their experience as supervisors ranged from one to eight years.

Question #9. "Have you ever had any formal training in cost estimation?" None of the eleven individuals indicated they had any such training.

Question #10. "Have you ever estimated the software cost of other projects?" Six had estimated software cost on other projects. Five had never estimated software cost on another project.

Discussion

The data indicates that there is a wide spectrum of experience among the respondents. There were some who had supervised programmers, had considerable software education, and had estimated software costs on other projects. On the other hand there were some individuals who had no programming experience, no software education, and no previous experience in estimating software cost.

Only four individuals had the type of work experience one might expect is necessary to determine the difficulty of a software module or to estimate the number of instructions in a module. Only three individuals had the level of software education one might expect is necessary to design a major new software system. Only one individual had both the work experience and education which might enable him to analyze a major software package in sufficient depth and to reasonably determine the resources required for software development.

In summary, it appears reasonable to reject the initial hypothesis. It appears that there is a wide variance in backgrounds with no common level of work or educational experience among software cost estimators at ESD.

Finding #3 - Software Cost Drivers

Hypothesis

There are a large number of factors which influence or drive the cost of software. Due to the nature of weapon system procurement, many of these factors may be unknown prior to contract award. For example, prior to contract award the type of computer may be an unknown factor.

Despite these unknown factors, a software cost estimator is still required to estimate software cost. To do this he must make a number of assumptions concerning the unknown factors. An hypothesis was formed to examine what factors are commonly unknown. The hypothesis was, "The software cost estimate is made when most of the software cost drivers are known." The ten factors or cost drivers examined were selected from the SDC study which indicated that these ten factors had a major influence upon cost (Nelson, 1967).

Sources Of Data

Two questions were formulated. The first addressed the level of knowledge of the ten factors prior to contract award. Data for this question came from the following ten program offices: 485L, 427M, TERPE, TIPI II, LORAN, AFEES, OASIS, A4BNCP, Pave Paws, and Cobra Dane.

A second question examined how well known these same ten factors were known after contract award. Data for this question was limited to the following six programs: 485L, TIPI II, LORAN, AFEES, Pave Paws, and Cobra Dane.

Questions And Responses

Two questions were formulated to examine the level of knowledge of ten factors prior to and after contract award.

Question #11. "At the time the program office made its first formal (e.g. Program Management Plan) estimate of software development costs, how well known were the following factors?"

a. Type of computer (e.g. UYK-7, UNIVAC 1108, etc.)
__Definitely known __Generally known __Slightly known __Unknown

b. Configuration and types of peripherals
__Definitely known __Generally known __Slightly known __Unknown

c. Memory and storage size
__Definitely known __Generally known __Slightly known __Unknown

d. Operating system
__Definitely known __Generally known __Slightly known __Unknown

e. Compiler and/or assembler
__Definitely known __Generally known __Slightly known __Unknown

f. Number and type of interfaces
__Definitely known __Generally known __Slightly known __Unknown

g. Number of input message types
__Definitely known __Generally known __Slightly known __Unknown

h. Number of output message types
__Definitely known __Generally known __Slightly known __Unknown

i. Response time requirements
__Definitely known __Generally known __Slightly known __Unknown

j. Number of Computer Program Configuration Items (CPCIs)
__Definitely known __Generally known __Slightly known __Unknown

To analyze the responses, different weights were assigned to indicate the extent to which a factor was known. The following weights were assigned:

Definitely known	- 10
Generally known	- 7
Slightly known	- 3
Unknown	- 0

With these weights assigned, it was easy to judge how well the various factors were known for each program. Again, each of the ten factors was assigned a weight between 0 and 10. Therefore, a maximum score of 100 would indicate that all ten factors were definitely known.

The responses indicated quite a range between programs. Some programs were follow-on developments with most of the factors well known. Other programs were at the other end of the spectrum. One program had a total score of 97 while another achieved a score of only 10. The mean score of the ten programs was 65.2 with a median of 68.

Thus, an estimator whose program had a score of 10 had a great many uncertainties facing him when a cost estimate was made. On the other hand, the estimator whose program scored 97 had only limited uncertainty facing him.

Examination of the individual factors yielded some interesting results. The least known factor was the operating system which can have a major impact upon software cost. The best known factor was the compiler. This is probably due to the fact that a JOVIAL compiler is frequently specified for most ESD programs.

Question #12. This question examined the same factors. However, the lead part of the question was, "At the time of contract award, how well known were the following factors?" Again, data was only obtained for six programs.

Using the same weighting system, the programs which answered both question 11 and 12 were examined. A marked improvement in the level of knowledge was obvious. Prior to contract award the six programs had a range of 10 to 97 with a mean value of 62.33. After contract award, the six programs had a range from 82 to 100 with a mean value of 86.83. As expected, once a winning contractor was selected, the factors which influence software costs were better defined.

Discussion

It is apparent from the wide spread of responses that the level of knowledge of the ten factors is dependent upon the nature of the acquisition. Those programs which are acquiring software for an existing data automation system have few uncertainties to address. On the other hand, those programs for which little is known about the system face many uncertainties.

The mean score of 65.2 prior to award indicates that for most programs many of the factors are known to some extent. As expected, the award of a contract greatly increases the level of knowledge of these factors.

In summary, the extent of knowledge of the ten factors is dependent upon the nature of the program. The level of knowledge varies greatly between programs at ESD.

Finding #4 - Management Reserve

Hypothesis

In light of the many uncertainties surrounding a software cost estimate, it seems reasonable to assume that program managers would budget additional funds for software as a management reserve. The following hypothesis was formulated to examine the nature and size of that management reserve: "A management reserve for software is maintained."

Sources Of Data

Only eight of the twelve individuals interviewed were familiar with the concept of a management reserve. Programs for which responses were obtained are: 485L, 427M, TERPE, TIPI II, LORAN, AWACS, Pave Paws, and Cobra Dane.

Questions and Responses

Six questions were administered.

Question #13. "Are you familiar with the concept of a "management reserve?" Eight individuals were familiar with the concept. The following five questions were then administered to these eight individuals.

Question #14. "Did the program office establish a management reserve for software?" Seven of the eight individuals indicated that their programs had established such a reserve. Sometimes this reserve was created by the contractor within his own budget. However, more frequently the reserve was created by the program office with funds not placed on any particular contract. Most of the respondents indicated that the reserve was not identified as such to prevent its loss during a budget cut.

Question #15. "If yes, was the management reserve left in the program until the contract was essentially complete?" This question was poorly worded. It applied only to programs which were complete and it assumed that the management reserve was not used. No responses were collected.

Question #16. "Do you feel that a software management reserve should be established for each major software acquisition?" Seven of the respondents strongly agreed with this concept. One disagreed feeling that such reserves would tie up funds required for other programs.

Question #17. "Do you feel that such a management reserve would be approved during the budget process at HQ AFSC and HQ USAF?" Four respondents felt that such a reserve would be approved in light of the frequent overruns in software acquisition. Three others felt that any reserve identified as such would not be approved.

Question #18. "What size of management reserve (as a percentage of the estimated software cost) do you feel a program similar to yours should budget?" The responses ranged from 5% to 50% with a mean of 18%.

Discussion

It is apparent from the data that most programs do establish management reserves. However, the reserve may not be explicitly identified as a reserve due to budgetary pressures.

Finding #5 - Independent Estimates

Hypothesis

A program manager sometimes tries to reduce the uncertainty surrounding a software cost estimate by having an outside agency prepare an independent estimate. At ESD these independent estimates are normally performed by the ESD Cost Analysis Division.

An hypothesis was formed to examine whether the independent estimate prepared for software was truly independent. The hypothesis was, "The independent cost estimate provided by an outside agency is truly independent."

Sources Of Data

Only five individuals were familiar with the independent cost estimate prepared for their program. Data were gathered for the following programs: 485L, TERPE, 427M, AFEES, and AABNCP.

Questions And Responses

Six questions were formulated to address this area. Unfortunately, data was only available for three of the six questions.

Question #19. "Was an independent cost estimate prepared for software?" Again, only five individuals were aware of an independent cost estimate prepared on their program.

Question #20. "What was the independent estimate?" This data was not readily available. Also, any figure which could have been obtained would have been difficult to assess. Often the independent estimate is made years before contract award. To compare this independent estimate with the current estimate would not be meaningful due to the many changes which might have occurred in the program. To compare the independent estimate to the program office estimate at that time was attempted. However, neither the program office files nor the files of the Cost Analysis Division had this information readily available.

Question #21. "What was the program office estimate at that time?" Again, this information was not readily available.

Question #22. "Which estimate do you feel is more accurate?" Again, no responses were collected since the relative estimates were not available.

Question #23. "Did you provide the independent estimator with your estimate of the number of instructions?" All five program offices which had independent estimates indicated that they had provided the independent estimator with their estimate of the number of instructions.

Question #24. "Briefly describe the type of information you provided to the independent estimator." All five of the individuals indicated that the independent estimators were given a complete description of how the program office estimate was arrived at. All assumptions and parameters were detailed to the independent estimators.

Discussion

Informal discussions were held with a number of the analysts from the ESD Cost Analysis Division. These discussions indicated that while different techniques were used to independently estimate software cost, all of the techniques relied heavily upon the estimated number of instructions since this software characteristic was generally available from the program office. While the independent estimators try to insure that an adequate engineering estimate is made by the program office of the number of instructions, they do not have the capability or the resources to independently estimate the number of instructions.

Therefore, since both the program office estimate and the independent estimate rely upon the same estimate of the number of instructions, it appears reasonable to reject the hypothesis. Unless the independent estimator can obtain an independent estimate of the number of instructions for use in his cost analysis, then the estimates he provides cannot be considered truly independent.

Finding #6 - Challenges of Estimates

Hypothesis

Since software cost estimating is a difficult and uncertain process, one might expect that software estimates are frequently challenged as to their accuracy. On the other hand, the lack of a reliable cost estimating technique may prevent any serious challenge of the initial software estimate. An hypothesis was formed to examine whether program office estimates or contractor estimates are challenged. The hypothesis was, "Software estimates are challenged."

Sources Of Data

Data were obtained for the following eight program offices: 485L, TERPE, 427M, TIPI II, AFEES, OASIS, AABNCP, and Pave Paws.

Questions And Responses

Two sets of questions were prepared. The first set addressed challenges of the program office estimate. Eight responses were obtained for this set. A second set addressed challenges of contractor's estimates. Due to poor question wording, no responses were collected for this question set.

Question #25. "Was the accuracy of the SPO's software cost estimate challenged by anyone?" The responses indicated that only one of the eight SPO's had their estimate challenged.

Question #26. "If yes, by whom?" The one challenge of a program office estimate was made by the Cost Analysis Division.

Question #27. "Was the accuracy of the contractor's software cost estimate challenged?" This question was poorly worded since the nature of the contractual relationship between the Air Force and the contractor requires such challenges.

Discussion

It is apparent from the responses that most program office estimates are not challenged. There are a number of reasons for this. First, many of the program office estimates are prepared with staff assistance from the Cost Analysis Division. Second, for an outside agency to question a program office estimate, they must have both a good understanding of the nature of the software acquisition and a good technique for estimating software costs. Without these, one must generally accept the program office cost estimate.

Finding #7 - Confidence Intervals

Hypothesis

When an estimator predicts a future cost, he does not expect the actual cost to fall exactly at that point estimate. Instead, he recognizes that a range of values are possible and selects a value within the range of possible values. He might select the most likely or mean value within the range. On the other hand, he may be conservative and select a value at the high end of the range.

Since different estimators might be using different points within the range of possible values, an hypothesis was formed to examine what the range was and what point in the range was being selected. The hypothesis was, "The confidence intervals which software cost estimators place on their estimates are narrow and uniform.

Sources Of Data

Since the questions were not applicable to those programs which were complete or nearly complete, the response was limited. Also, some individuals did not wish to guess at how high or how low software costs might be. This may have been due to the wording of the question.

Data was collected from five program offices. Because of the somewhat sensitive nature of some of the responses, the programs are not identified.

Questions And Responses

Question #29. "What was the program office estimate of software cost prior to contract award?" Rather than identify the sources of each response, Table 3 displays the responses for the next five questions in an anonymous fashion. Also, no specific dollar figures are given. Instead, each of the answers is given in terms relative to the program office estimate.

PROGRAM	PROGRAM OFFICE ESTIMATE	CONTRACTOR ESTIMATE	POSSIBLE LOW	POSSIBLE HIGH	PERCENT COMPLETE
Program A	A	0.26A	1.20A	1.20A	60
Program B	B	N/A	0.6B	B	0
Program C	C	C	0.84C	?	20
Program D	D	D	D	1.40D	0
Program E	E	E	E	1.20E	0

Table 3
Range of Estimates

Question #30. "What was the contractor's estimate of software cost in his proposal?" Again, the responses are shown in Table 3. Note that for three programs (i.e. C, D, and E), the program office estimate and the contractor estimate are the same. After contract negotiations, these three programs felt that the initial contractor's estimate was a reasonable value and altered their program office estimate to match it.

Note that for Program A, the contractor's estimate was only 26% of the program office estimate. While a buy-in or a misunderstanding is indicated, the contract award was still made at a price which was 26% of the program office estimate. Without strong evidence of intentional underbidding, the pressure is upon the contracting officer to award to the lowest bidder. The lack of a reliable method for verifying a software cost estimate may have been the reason that sufficient evidence could not be developed.

Program B had not yet awarded a contract.

Question #31. "How low do you believe the actual cost of software might eventually be?" Again, the responses are indicated in Table 3. Note that two programs (i.e., D and E) consider the program office estimate to be the lower bound of the range. On the other hand, Program B considers the program office estimate to be the upper bound of the range.

Program A's response indicates that the program office expects the contractor to complete at 120% of the initial program office estimate. This is 460% higher than the initial contract award. The program office is reasonably confident of this cost to complete figure since a majority of the contractual effort is complete.

Question #32. "How high do you believe software costs might eventually be?" Again, the data is presented in Table 3. The respondent for Program C did not wish to address this question.

Question #33. "What percentage of the contract period is complete?" This data is also presented in Table 3.

Discussion

It is apparent from the data that software cost estimators do not select the same point within the range of possible costs. Program B selected a value at the high end of the range, Program C selected a middle value, and Programs D and E selected values at the low end of the range. While the limited data precludes any conclusion concerning the confidence interval surrounding the estimate, the selection of widely different points within the range of possible costs provides some interesting information. A given software cost estimate at ESD might represent the lowest possible cost, the most likely cost, or the highest possible cost depending upon the particular program. Apparently no guideline exists which directs the selection of a certain point in the range.

Finding #8 - Software Development Phases

Hypothesis

In preparing a software cost estimate, an analyst must address each of the software development phases or activities which are applicable to his program. While most software acquisitions include the common development phases or designing, coding, and debugging a computer program, there are many other development phases. A software acquisition may include an analysis of requirements prior to software design. Maintenance of software is another typical development phase.

Most of the software cost estimating techniques that have been developed are based on estimating the cost to design, code, and debug software since these phases are common to all software developments. An hypothesis was formed to examine which phases were most frequently contained in the software acquisitions at ESD. If a program includes more than the common three phases of design, code, and debug, then the use of only a single cost estimating technique which addresses only these three phases would not be sufficient. Other techniques would have to be used to estimate the cost of the other development phases. Also, if a cost researcher is trying to develop a useful cost estimating technique, he should be aware of the various phases for which a cost must be estimated. To examine which phases were included in the programs at ESD, the following hypothesis was formed: "The phases of software development included in ESD programs are common."

Sources Of Data

Data were collected from the following twelve program offices: 485L, TERPE, AFSATCOM, 427M, TIPI II, LORAN, AWACS, AFEES, OASIS, AABNCP, Pave Paws, and Cobra Dane.

Questions And Responses

Only one question was used to examine this hypothesis. The question sought to identify which of sixteen software development phases were applicable to a particular contract.

Question #34. "Which of the following contractor tasks were included in the program office estimate of software cost?"

a.	Analyze user requirement	___Yes	___No
b.	Prepare system specification	___Yes	___No
c.	Define system interfaces	___Yes	___No
d.	Design the data base	___Yes	___No
e.	Develop program test plans	___Yes	___No
f.	Specify all input and output message formats	___Yes	___No
g.	Design and flow chart each computer program component	___Yes	___No
h.	Write coded program statements	___Yes	___No
i.	Compile and check program code	___Yes	___No
j.	Plan and run functional test of each program	___Yes	___No
k.	Plan and conduct demonstration test	___Yes	___No
l.	Train user personnel	___Yes	___No
m.	Conduct demonstration test	___Yes	___No
n.	Assist in operational shakedown	___Yes	___No
o.	Develop software maintenance plan	___Yes	___No
p.	Maintain software after delivery	___Yes	___No

The responses to this question are depicted in Figure 3. Some of the programs include all sixteen software development phases. One included only ten of the phases. On the average, more than fourteen of the phases were included in a particular contract.

SOFTWARE DEVELOPMENT PHASES	NUMBER OF CONTRACTS WHICH INCLUDE A PARTICULAR PHASE
Analyze user requirement	6
Prepare system specification	9
Define system interface	10
Design the data base	11
Develop program test plans	12
Specify all input and output message formats	10
Design and flow chart each computer program component	12
Write coded program statements	12
Compile and check program code	12
Plan and run functional test of each program	12
Plan and conduct integration test	12
Train user personnel	11
Conduct demonstration test	12
Assist in operational shakedown	12
Develop software maintenance plan	6
Maintain software after delivery	6

Software Development Phases

Figure 3

Note that only six programs included the following three phases: analyze user requirement, develop software maintenance plan, and maintain software after delivery. While these activities are performed by nearly all of the programs, they are frequently performed under a separate contractual effort.

Discussion

It is apparent from the data the while most of the contracts have many common cost elements, some contracts include or exclude certain major software development phases. When reviewing a cost estimate at ESD, it is necessary to examine which of the sixteen phases are included in that estimate since there is only limited commonality between contracts. One cannot assume that a given software cost estimate addresses the same software development phases.

Finding #9 - Cost Proposals

Hypothesis

A contractor's cost proposal can provide significant management information to a program office. If a contractor's cost proposal includes the method used to estimate software cost, a program office might be better able to determine if a contractor fully understands the nature of the software task. Also, if the contractor's estimate is based upon a certain programmer productivity figure, this figure can provide the program office a method of judging the status of software development. For example, if a contractor estimated that his programmers would produce eight instructions per day during the term of the contract, then any significant deviation from this figure would indicate that the total man-hours required is changing from the initial estimate.

An hypothesis was formed to determine the type of software cost estimating information which is provided in a cost proposal. The hypothesis was, "The contractor's cost proposal indicates the method used to predict the cost of software."

Sources Of Data

Seven program offices were able to provide data. The are: AFSATCOM, 427M, TIPI II, LORAN, AWACS, AFEES, and Cobra Dane. Other programs were either not on contract or the individual was not familiar with the contractor's cost proposal.

Questions And Responses

Question #35. "Did the contractor's cost proposal indicate how the cost of software was estimated?" Four individuals indicated that the method for estimating software cost was explained in the cost proposal. The other three individuals indicated that there was no supporting rationale.

Question #36. "If yes, briefly describe the method the contractor used to support his software cost estimate." Three contractors first estimated the number of required instructions. Then, they applied a programmer productivity factor to determine the amount of direct labor required. In one case, the contractor's productivity factor was supported by data from a previous contract.

A fourth contractor estimated cost by performing an engineering cost analysis. The software was divided into modules and each module was estimated in terms of the number of man-hours required. No estimate of the number of instructions was made.

Discussion

It is apparent that the inclusion of the software cost estimating method in the cost proposal is somewhat random. When the method is provided, it does supply significant information. For example, the estimate of the number of instructions furnished by three contractors could be checked against the program office estimate. Any significant difference would indicate the need for additional technical discussions. Also, by providing productivity factors, the contractors provided the Air Force with an additional method for monitoring the status of software development.

The three programs, which did not receive any supporting data, run the risk of awarding to a contractor at an unreasonably low price. The problems created by such awards were discussed in Chapter II.

Finding #10 - Contractor Bids

Hypothesis

When a contractor estimates the cost of software in a proposal, a number of factors can affect the bid price. His understanding of the requirements, his labor rates, and his pricing policy are major factors. However, his method of estimating software cost is also an important factor. While the effects of these factors are difficult to separate, it appears reasonable to assume that understanding, labor rates, and pricing policies should not result in radically different prices for software. Instead, any radical difference is more likely due to different methods for estimating software cost.

An hypothesis was formed to determine if radical differences in the bid price for software do occur. While the causes of these differences are uncertain, an assertion is made that a primary factor is the different methods used to estimate software costs. The hypothesis was, "Contractor's estimates in response to a software Request For Proposals (RFP) do not vary widely."

Sources Of Data

Data could only be obtained on three programs at ESD. Due to the sensitive nature of contractors' pricing data, the specific programs involved are not identified.

Questions And Responses

Three questions were formulated which sought to identify only the highest and lowest bidder and determine the range of the bids. However, since data was available for only three programs, all of the bids for a program are examined to provide additional insight.

Question #37. "Was the contract for software awarded competitively?" All three of the contracts were awarded competitively and involved both hardware and software. Only the bid price for the software line item was examined.

Questions #38 and #39. These questions only sought to identify the highest and lowest bidders. Instead the entire spectrum of bids is examined for each of the three programs.

The first program had seven bidders, all major defense contractors. The Air Force estimate of software cost was $150,000. One contractor estimated the software cost at one-third of this figure while another estimated the cost at almost six times higher than the government estimate. The other five bidders were distributed evenly between these two extremes. Note that the highest bid was 17 times higher than the lowest bid. The contract was awarded to a contractor whose software estimate happened to match the Air Force estimate. (Note: This effort consisted of both hardware and software. Therefore, the bid price for software was not the only factor in making the award.) At the completion of the contract, the actual software cost was fairly close to the government's original estimate.

A second program had only three bidders with a much smaller range of bids. If we call the government's initial estimate "X", then we can examine the bid prices in relation to the government estimate. The three bids were 66% of X, 84% of X and 126% of X respectively. While these bids bracket the government estimate, there is still almost a two-to-one ratio between the highest and the lowest bidders. Since this was a multi-million dollar software effort, the difference was quite substantial.

Data on a third program was obtained from a briefing given by the chief of the ESD Cost Analysis Division (Grimm, 1976). The data provides the responses of five contractors who were bidding against the same statement of work. The data includes the contractors' estimates of the number of instructions.

CONTRACTOR	INSTRUCTIONS	COST
A	153,000	$2,800,000
B	282,000	$2,500,000
C	400,000	$4,600,000
D	735,000	$4,500,000
E	766,000	$2,100,000

Note that the contractor with the largest estimate of software size (i.e. Contractor E) bid the lowest software price. Note also the range in the sizing and in the cost. The sizes range from a low of 153,000 instructions to a figure five times as big. The cost range is narrower but still represents more than a two-to-one ratio between the highest and the lowest bidder.

Discussion

Again, the reasons for the differences are uncertain. It can be seen that there is a wide range not only in the pricing of software but also in sizing software. How much of this cost variance can be attributed to the method used in estimating software cost cannot be determined. Admittedly, some or even most of the difference might be due to misunderstandings of the requirements. However, if we look at bidders D and E from the last example, we can see that despite reaching similar conclusions concerning the size of the effort, the dollar values assigned to their efforts are quite disparate.

It seems reasonable, even in light of the limited data, to refute the hypothesis. Contractor's estimates of software costs do appear to vary quite significantly when responding to the same specifications and the same statement of work.

Finding #11 - Availability of Data

Hypothesis

Finding #2 described the various types of software cost estimating techniques which are in use at ESD. These techniques rely heavily upon such parameters as cost per instruction and programmer productivity. Since the accuracy of these parameters is important, an hypothesis was formed to examine whether the contracts at ESD were collecting the type of software cost data necessary to verify these parameters. While verifying these parameters might have only limited utility for the current programs, it could serve to guide future software cost estimating efforts. The hypothesis was, "The type of software cost data currently being collected on software contracts will support the types of cost estimating methods currently being used."

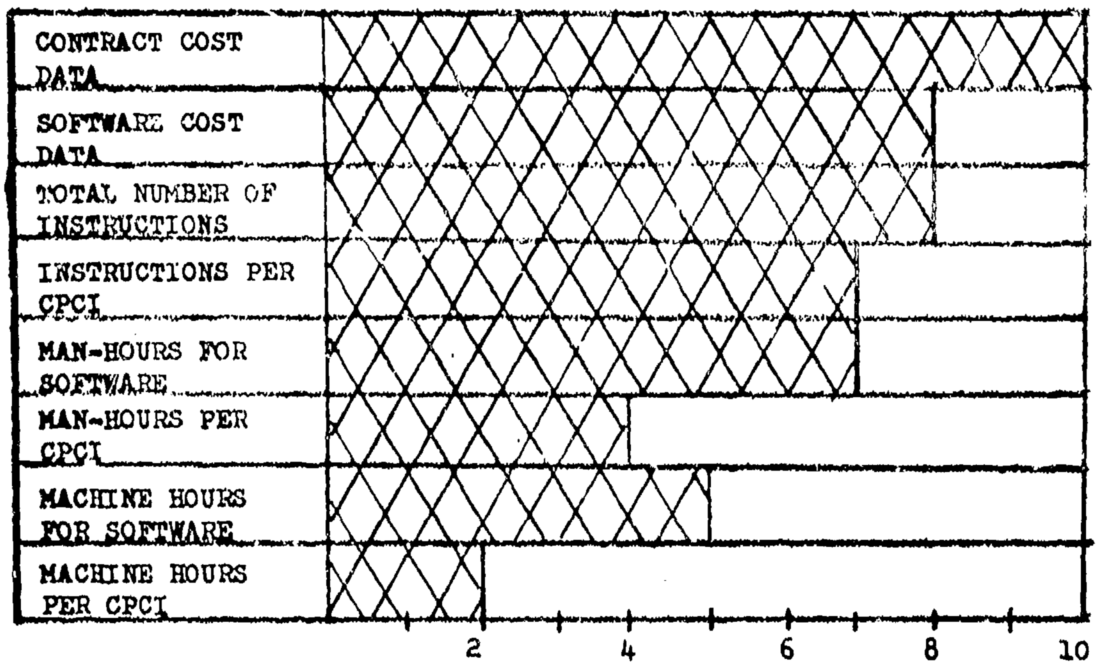
Sources Of Data

Data was obtained for the following eleven program offices: 485L, 427M, TERPE, AFSATCOM, TIPI II, LORAN, AWACS, AFEES, AABNCP, Pave Paws, and Cobra Dane.

Questions And Responses

Eight questions were asked to determine what elements of information were being collected by the eleven programs. The responses to each element of information are summarized in Figure 4.

Question #40. "Does/will your contract call for the reporting of cost data?" Ten of the eleven programs indicated they do receive contract cost data.



Type of Information

Number of Programs for which Information is Available (Eleven Programs Interviewed)

Availability of Data

Figure 4

Question #41. "Does/will the contractor report the cost of software as a separately identified item?" Eight of the eleven indicated that software costs are separately identified. This is significant. Major programs at the Aeronautical Systems Division (ASD) such as the B-1 do not separately identify software in their cost reports. At ESD, however, the practice is quite common.

Question #42. "Does/will the contractor report the total number of computer instructions?" Again, eight of the eleven programs receive this data.

Question #43. "Does/will the contractor report the total number of instructions for each Computer Program Configuration Item (CPCI)?" (Note: A CPCI is a software module which includes a major segment of the software being acquired. For example, a compiler or an operational flight program might be CPCI's. If data is available to the CPCI level, an estimator might be able to draw analogies between similar modules for different programs). Seven of the eleven programs receive this level of data.

Question #44. "Does/will the contractor report the total number of direct man-hours charged to software?" This data is required to determine programmer productivity factors. Seven of the eleven programs collect this data.

Question #45. "Does/will the contractor report the total number of direct man-hours charged to each CPCI?" This information would be necessary to determine if different productivities existed for different types of software modules. Only four programs receive this data.

Question #46. "Does/will the contractor report the total number of machine hours for software development and test?" Computer time can be a significant portion of software cost. While some of the ESD programs provide this resource as a government furnished item, others are charged for machine hours by the contracto . To be able to compare cost data from two programs, an analys : would have to know whether or not this cost is included in the total software cost. However, only five of the eleven programs collect this data.

Question #47. "Does/will the contractor report the total number of machine hours required for the development and test of each CPCI?" Only two programs received this level of detail.

Discussion

The collection of data by programs at ESD is far from uniform. Some programs collect all of the above data elements while others collect only a few or even none of them. To develop a reliable software cost estimating data base, controlled and well defined data should be collected from all programs.

Many of the individuals reported that while the data was not formally collected, it might be obtainable from the contractor. However, unless a specific and well defined set of software cost elements are contractually required and formally reported for all programs, the development of a comprehensive and productive software cost data base is extremely difficult.

In general, it appears that many programs at ESD are collecting the types of data necessary to develop a software cost estimating data base. Unfortunately, some serious gaps exist in the data collected. There is no common format or definition of variables for collecting this data. Also, there is no common repository for the data that is being collected. Unless this data is uniformly collected and analyzed for many programs, the utility of the currently collected data in developing better cost estimating techniques is minimal.

Finding #12 - Software Status

Hypothesis

Because of the lack of well defined technical milestones for software development, managers have difficulty in assessing the status of software. Nevertheless, the status must be determined for management purposes such as determining the amount of progress payments to allow a contractor. The status of software is frequently given by a single parameter - the percent complete. While the validity of such a parameter may be questioned, it is in widespread use at ESD.

An hypothesis was formed to try to determine how the percent complete of software was computed. The hypothesis was, "A primary management indicator of software status is the comparison of budgeted cost and actual cost. The percent complete of the software task is equal to the ratio of actual cost to budgeted cost." In Chapter II the problem of using this ratio as a status measurement was discussed. If the software cost estimate is in error, then a manager may well be monitoring financial activity instead of technical progress.

Sources Of Data

While it was possible to obtain data concerning the actual and budgeted cost of software for many programs, it was not possible to readily collect data concerning what percent complete the software was judged at any specific time. While such data may exist in the contracting officer's files, the individuals interviewed were not able to provide this information. Therefore, no data was collected concerning this hypothesis.

Finding #13 - Changes in Software Cost Estimates

Hypothesis

After making an initial estimate of software cost, a manager will frequently obtain additional information which requires a revision of the cost estimate. An hypothesis was formed to examine when and how these changes occur during a program's lifetime. The hypothesis was that the software estimate did not change.

Sources Of Data

Data was gathered from ten programs at ESD. Five of these programs were complete while the other five are still ongoing. Due to the sensitive nature of the data, the individual programs are not identified.

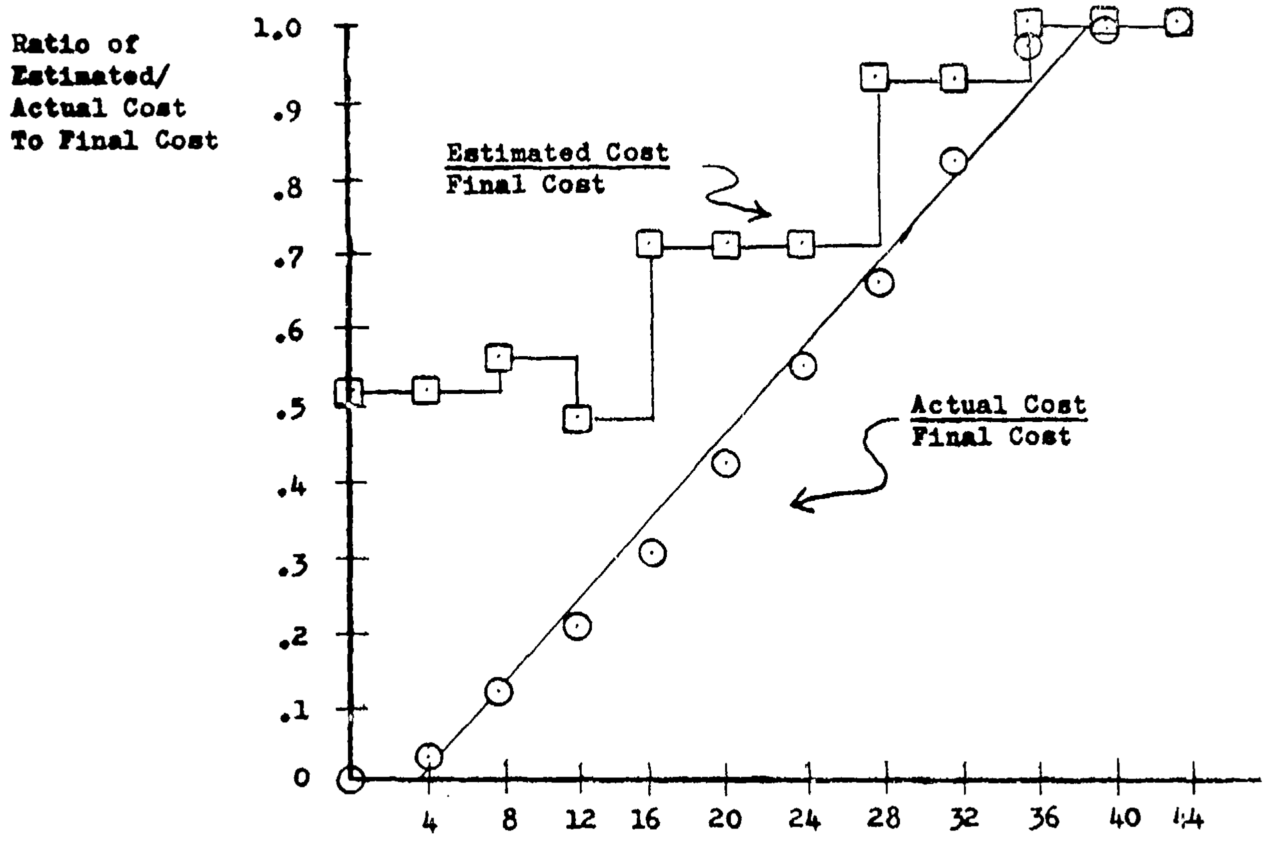
Questions And Responses

Initially, a set of eleven questions was formed to address this hypothesis. The questions appear in Appendix B as Questions 51 through 61. Sufficient response was not obtained on these questions for two reasons. First, the questions assumed that each program office maintained an historical record of the initial software cost estimate and of the changes to the estimate during the lifetime of the program. Such data was not consistently recorded by the program offices. Second, the questions were poorly worded. For example, one question asked for the cost estimate at the time of the Critical Design Review (CDR). It was thought that the CDR was a single point in time common to most programs. Unfortunately, while the CDR is common to most programs, it sometimes covers a considerable period of time and is not a single point in time. Therefore, the software estimate during the period of CDR sometimes changes quite significantly.

In light of the unavailability of responses to the eleven questions, an alternative source of data was used. Many of the contracts at ESD receive periodic reports from contractors concerning the progress of a contract. One of these reports is the Cost Performance Report (CPR). The CPR reports the contractor's progress against an initial financial plan. A budget or financial plan is established for each major task at the start of the contract. As the contract progresses, the contractor reports his actual expenditures and explains any variances between his estimated expenditure and his actual expenditure.

While the CPR contains many items of information, only two were relevant to this hypothesis. The first item is the contractor's actual cost expended for software. This actual cost includes direct labor and overhead. The second item is the contractor's estimate of total software cost. As the contract progresses, the contractor reports both his increasing actual cost and any changes which he may make to his estimate of the total software cost.

Completed Programs. We can first look at the data which was obtained from the five programs whose contracts are completed. Figure 5 depicts the type of information gathered from Program A.



Number of Months After Contract Award

PROGRAM A - Cost Performance Data

Figure 5

The horizontal axis of the graph depicts the number of months elapsed after contract award. The vertical axis depicts the ratio of estimated or actual cost to final cost. Two plots are made. One is the ratio of actual cost to final cost over time. The second is the ratio of the estimated cost to final cost over time. For example, Figure 5 indicates that the initial software cost estimate was only 52% of the final software cost. Also, the graph indicates that the actual cost equaled the initial estimate when only 22 of the 44 months of contractual effort were complete.

Some interesting observations can be made from Figure 5 which is representative of the data obtained for all five completed programs. First, one notes that the ratio of actual cost to final cost is essentially linear. The implication is that the level of resources applied to software is level throughout the period. Note that for Program A, the contractor probably sized the initial software team based on his initial estimate of software cost. However, this estimate was only 52% of the final cost. When the contractor began to realize that his initial estimate was faulty, he did not take any action to significantly increase the amount of resources applied to software.

Two reasons can be postulated for the contractor's failure to increase software resources. First, increasing resources adds significant cost and problems to the project. The existing software team must normally stop software production while new staff members are trained and oriented to the new project. Second, the contractor does not quickly realize that his initial estimate was grossly wrong. For example, after 16 months the Program A contractor finally raised his estimate of software cost. However, the increased value was only 72% of final cost and was not significantly higher than the original estimate. Later, in the 28th month, the contractor again raised his estimate of software cost. He now knew that his initial estimate was grossly wrong, but it was too late to make any significant improvement in schedule performance by adding more resources.

In essence, it appears that the contractor has the attitude that good times are just around the corner. While his estimate might be grossly wrong, he learns of this gross error only slowly. The increases in the estimate never appear to be large enough to merit the application of additional software resources. The net effect is that Program A software, which was expected to take 22 months and cost a certain amount, wound up taking 44 months and costing almost twice as much.

The impact of this attitude is twofold. There is of course the additional cost of software. However, this additional cost might be minor in impact compared to the indirect cost of a 22 month schedule slip. Such a slip can generate indirect costs which well exceed the additional cost of software.

A second observation can be made from the Figure 5 data. Note that the estimated cost of software appears to change in irregular steps. Again, two reasons can be postulated for this. First the contractor has no method for easily updating his initial cost estimate. He does not generally have a cost model which can be quickly updated to take into account current performance. Instead, reestimating is a major task. The software production is halted while a new software development plan is formulated and resources are estimated against this new plan. This type of major reestimating effort can compound the problem of rising software cost. Therefore, contractors tend to avoid reestimating until it is blatantly obvious that the initial plan is faulty.

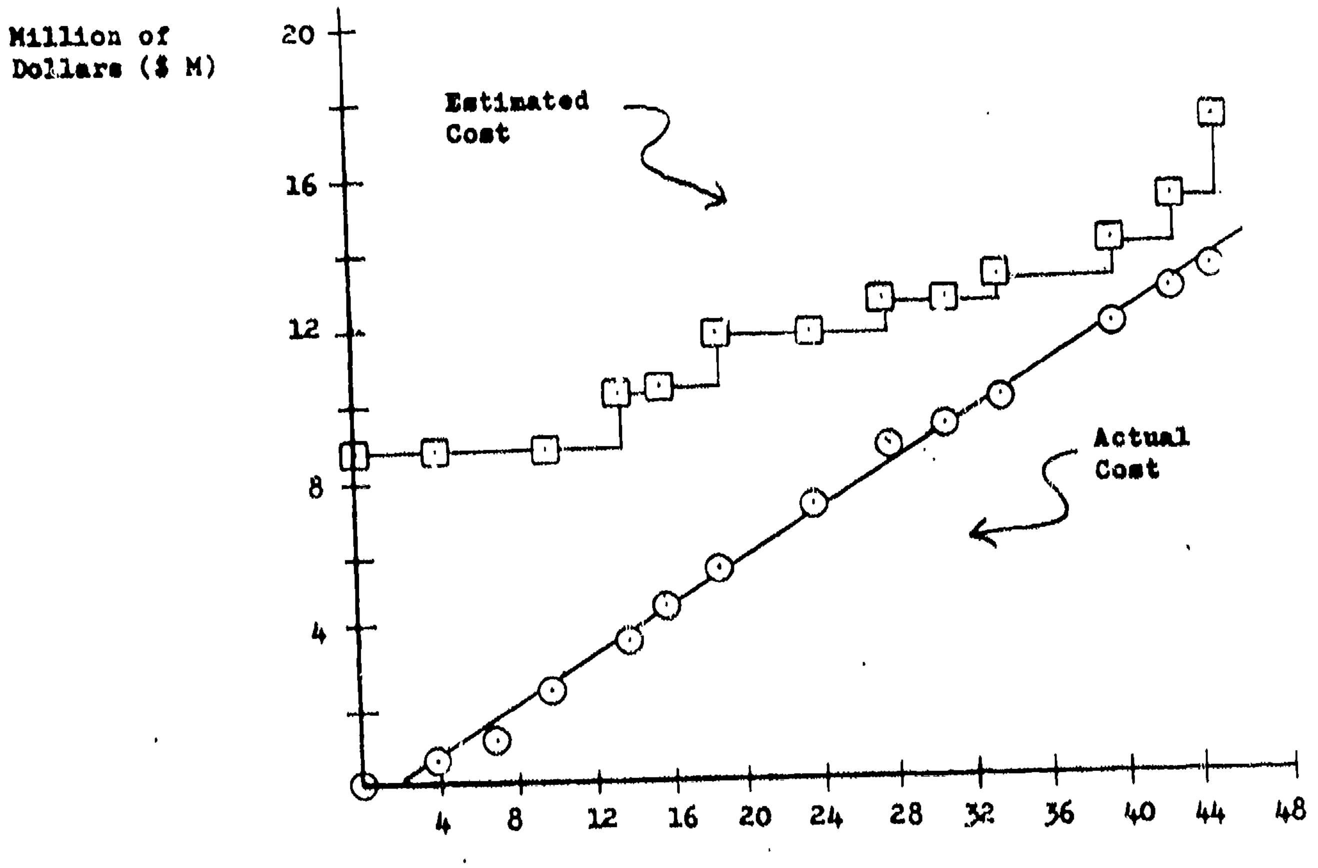
The second reason for the step-like behavior of the estimated cost data is that the contractor frequently is uncertain that his current estimate is in error until his actual cost approaches his estimated cost and the software is still incomplete. Note that for Program A the actual cost almost equaled the estimated cost in the 27th month finally forcing the contractor to realize the need for reestimation.

A third observation can be made by looking at the changes in the software estimate during the early months of the contract. One might expect that after the software team has been assembled and working for three to six months that they would then have a valid conception of the size of the software. However, looking at Figure 5, we can note that the estimate actually dips in the first months - increasing the estimating error. Not until 16 months into the program is a significant increase in the software cost recognized. Apparently, the first months of a contract are a period of optimism during which any indications of higher software costs are either not available or are ignored.

Appendix C contains the graphs of four other completed programs (Programs B, C, D, and E). While the initial cost estimate for these programs varied from 38% to 78% of final cost, the general pattern of all five programs is somewhat similar. Software costs are accumulated linearly; software estimates change in irregular steps; and no major change occurs in the estimate early in the program.

Uncompleted Programs. Having looked at how completed programs performed, we now turn to the five uncompleted programs. Figure 6 displays the data gathered for Program F.

Note that since the programs are still ongoing, no final cost of software is known. Therefore, the vertical axis of Figure 6 represents the dollar value of the estimated or actual cost of software. Note, for example,



Number of Months After Contract Award

PROGRAM F - Cost Performance Data

Figure 6

that the initial cost for software for Program F was about $8M. After 45 months of effort the estimated cost to complete has grown to $17M, more than double the original estimate.

Again, the same three observations can be made for the uncompleted programs that were made for the completed programs. For example, the actual cost are again being accumulated in a linear fashion. Also, the changes in software estimates appear to occur in irregular steps. And again, no major change in the software estimate is evident for the first 16 months of the contract.

Appendix D contains the data collected for both the completed and uncompleted programs graphed in the same fashion as Figure 6. The graphs indicate different values and different sizes of software errors. However, all ten of the programs tend to exhibit the same three properties that were observed for Program A and Program F.

Discussion

Again, the Cost Performance Report data appears to indicate a number of things about software cost estimates. The implications that these observations can have for management are important.

Linear Cost Accrual. The data indicate that none of the ten contractors ever significantly altered the size of the original software team. The contractor will normally keep the initially formed team working until the software is eventually completed.

This implies that any error in estimating software cost will eventually be translated into a software schedule slip. For example, in Program A the contractor originally estimated software to cost $1.7M and take 22 months. At the end of the contract, software cost had risen to $3.4M. Since the program was a large hardware and software development, the dollar increase in software had only a small impact. However, the software schedule was extended from 22 months to 44 months. The indirect cost of this schedule slip cannot be determined. However, it was quite evident from discussions with program office personnel that the schedule slip had a much greater impact than the increased cost of software.

Thus, if managers view the completion schedule of their program as a critical factor, then managers should be equally concerned with the estimation of software cost. Any error in the software cost estimate will probably be translated into schedule slippage.

Step Changes In Estimates. One might expect that if a software manager saw that a certain software module had been grossly underestimated, he might take action to revise his estimate of the remaining modules. In fact, the software manager does not appear to quickly react to any such indications. Only when the actual cost of software approaches the estimated cost does the software manager appear to be sufficiently motivated to stop producing software and to revise his software estimate. This type of behavior is demonstrated in the irregular stepwise changes in the software estimate.

Early Optimism. It also appears that software managers are quite optimistic during the early months of a contract. Again, one might expect that during the early months a more definite software development plan is established which accurately reflects the total effort required. Given much more time and resources than were available during the preparation of contract proposals, one might expect that the new plan during the early months would indicate the need to accurately revise resources.

In fact, such a revision of software estimates does not appear to occur during the early months of the contract. In many cases, the estimate of software cost actually drops and becomes worse. Not until quite late in the contract does the contractor slowly and gradually realize the error in his initial estimate.

The Good And The Bad. None of the five completed programs ever had the final cost of software lower than the initial estimate. The best performance was from two programs which exhibited cost growths of about 30%. If one considers that these estimates were made for a two year period during which inflation was quite high, then the accuracy of these two program estimates is remarkable. On the other hand, there are other programs whose software cost has doubled and continues to climb. In short, the range in program performance, as indicated by the initial software cost estimate, is quite broad. In light of the vastly different software cost estimating parameters and techniques in use, as presented in Finding #1, the wide disparity of program performance is not surprising.

VI. Recommendations

In 1976, DoD managers will be making decisions concerning the acquisition of an estimated $3 billion of software. Since cost is frequently the dominant criteria in these decisions, the capability to provide these managers with reliable and accurate software cost estimates is essential.

This research effort has provided some limited insight into the software cost estimating process at ESD. Based upon the research findings, there appear to be some major problem areas which inhibit the development of accurate and reliable software cost estimates. This chapter reviews these problem areas and recommends actions to improve the software cost estimating process.

While this research was limited to the ESD environment, the same types of problems may exist at other DoD software acquisition activities. Therefore, the adoption of these recommendations by other DoD agencies should also be considered.

Program Office Estimates - An Assessment

One part of the software cost estimating process which can be improved is the development of internal program office estimates of software cost. Before examining possible improvements, it is important to understand the nature of the problems estimators at ESD have in developing accurate and reliable estimates.

Error Sources. In estimating software cost, there are three possible sources of error. The first source of error is due to the element of chance in the future which makes cost a random variable. No matter how good an estimating technique might be, the random nature of future cost will always be a source of error. No action can be taken to completely eliminate this source of error.

A second source of error is the estimating technique itself. No current technique has been demonstrated to consistently produce reliable and accurate estimates. Current techniques fail to precisely and completely describe the relationship that exists between cost and other parameters. Until future research develops better techniques, this source of error will continue to trouble program office estimates.

There is a third source of error which may, in the writer's opinion, contribute more error than the other two sources. This third source is the non-uniform and unskilled application of a cost estimating technique. Regardless of how good a technique is, an accurate estimate may only be obtained if the technique is properly used. The research findings indicated that a number of problem areas exist which give rise to this type of error.

Problem Areas. For example, instead of using a common, well-defined method for estimating software cost, Finding #1 indicated that ESD estimators use a myriad of different and poorly defined techniques. Cost parameters vary widely. Managers find it difficult to compare estimates prepared by different techniques. They also find it difficult to judge the relative merits of the various techniques since not enough programs use the same methods and the same cost parameters

The many different techniques, coupled with some unskilled estimators, often can result in errors. In one case, an estimator input an estimate of 20,000 instructions into a cost model only to find later that the true number of instructions was 174,000. Another estimator interviewed did not understand the difference between source and object instructions. In a third case, a misunderstanding of terms resulted in a $33 million estimate for a software project probably a tenth of that size. These types of mistakes can result in more net error than any other source.

Other problems exist. Finding #7 indicated that a given estimate might represent either the lowest possible cost or the highest possible cost. Finding #8 indicated that all estimates do not address the same phases of software development.

Faced with these problems, managers seek outside help to bolster their confidence in an estimate. However, Finding #5 pointed out that the current independent estimates were strongly dependent on program office estimates of the number of instructions. The independent estimate is likely to suffer from the same sources of error as the program office estimate. Finding #6 determined that most software cost estimates are unchallenged. Managers cannot count on errors being detected by some outside agency.

One major cause of these problems was possibly identified by Finding #2. The software cost estimators at ESD do not possess a common, minimal level of work or educational experience which might qualify them to properly analyze a major software effort and estimate software cost. While a few were well qualified, others had almost no work or educational background in software development.

<u>An Assessment</u>. Based on the research findings, it appears reasonable to conclude that the current problems in the software cost estimating process at ESD are inhibiting the development of the accurate and reliable estimates which managers need to make good software management decisions.

<u>Program Office Estimates - A Recommendation</u>

One approach to minimizing the above problems is to have all ESD program offices utilize a common, sound, software cost estimating technique.

Based upon this research, it is recommended that ESD tentatively adopt a forthcoming software cost estimating addition to the RCA PRICE model as the standard ESD software cost estimating technique.

Objective Of The Recommendation. There is nothing that can be done to reduce the estimating error due to the random variable nature of cost. Likewise, until further cost research is performed, little can be done to reduce the error inherent in any of the current estimating techniques. However, much of the error in program office estimates may be caused by the problems identified in the research findings. The objective of recommending adoption of the RCA PRICE model as a standard technique is to address this type of error.

Advantages Of A Common Technique. Independent of what technique is adopted, the use of a common technique promises many improvements to the current software cost estimating process:

1. If estimates were produced from a common technique, managers could then reasonable compare two estimates in making a decision.

2. Selecting a well-founded technique would eliminate the use of many other techniques which are suspect in their predictive ability. The wide range of unsupported cost parameters in use would be narrowed.

3. A common technique could insure that all estimators addressed the same software cost elements and the same software development phases in their estimates. Each phase could be explicitly identified and estimated.

4. To lessen the problems caused by the disparate backgrounds of the estimators, a minimum amount of training in the use of a common technique might be required. In lieu of such training, a common technique could be well-defined and well-documented to minimize misinterpretations and errors in its application.

5. With all ESD programs using the same technique, a wealth of information would soon be available to determine whether the technique was successful and to determine how the technique might be improved. The current use of many different techniques inhibits this type of feedback.

6. The common technique could specify which point in the range of possible costs is to be selected. No longer would estimates range from the lowest possible to the highest possible cost.

7. Finally, adopting a single technique would aid the independent estimators at ESD. Instead of having to verify the reasonableness of many different and ill-defined techniques, the independent estimator could simply have to insure that the common method was properly applied. He might then spend more time and effort in insuring that the parameters input to the model were reasonable. Outside engineering assistance to verify the estimated number of instructions might be obtained.

The RCA PRICE Model. The specific technique recommended for tentative adoption is a forthcoming addition to the existing RCA PRICE model (RCA, 1975). The PRICE model was developed by RCA and has been marketed to a wide variety of industrial and DoD users. To date, the model has primarily been limited to the estimation of hardware cost. However, in the Fall of 1976, RCA plans to expand the model to include the capability to estimate the cost of software development.

The software cost estimating addition to the PRICE model is, to some extent, an adaptation of the Wolverton technique discussed in Chapter III. The technique first requires the estimator to divide the software into modules and to estimate the size of the individual modules. The design is required to produce modules no larger than 1000 instructions in size. The estimator is then asked to determine the type, complexity, and data storage requirements of each module as well as any similarity between the new module and past efforts.

After the modules have been classified, the model provides "cost per category of instruction" factors to estimate the cost of each module. These factors are based upon experiences at RCA for a particular category of instruction.

For each software development phase included in a particular program, the model insures that each phase is explicitly addressed. It also insures that the various elements of software cost such as direct labor and computer time are also explicitly addressed.

While the model is quite detailed, the information and judgments required to use the model are fairly simple once a detailed software design is available. The model is an on-line computer model which makes it simple for the estimator to input his information and judgments. It also makes it easy for an estimator to quickly determine the cost impact of proposed changes to the software parameters.

Despite the promising aspects of the RCA PRICE model, it has not yet demonstrated that it is any better than other techniques in consistently producing accurate and reliable cost estimates. However, the reason for recommending the PRICE model is not that it has proven to produce better estimates. Instead, the recommendation is made due to the workable nature of the model as well as some unique advantages it offers.

RCA PRICE Model - Advantages. The RCA PRICE model is the result of a well supported research effort by a major hardware and software contractor. It has been developed with the best of motives - profit. However, there are other reasons for recommending its use:

1. The RCA PRICE model will be easily available to all ESD program offices. The ESD Cost Analysis Division contracts with RCA for the right to use the model. Consultation services as well as a complete training program are also made available from RCA.

2. The PRICE model is now used to estimate hardware cost for all ESD programs. In a briefing by the chief of the Cost Analysis Division, the PRICE model was said to consistantly produce estimates within 10% of the actual cost (Grimm, 1976). While this same success might not be shared by the software addition to the model, the current success of the model as well as its reputation merit its tentative adoption.

3. While the model requires a detailed software design, the program offices at ESD have access to adequately qualified software personnel who can properly utilize the model. No sophisticated knowledge of cost estimating or of modeling is required.

4. One added advantage of the model is that it requires the software to be divided into 1000 instruction modules. By requiring this level of detail in the design, the model insures that the operational user's requirements have been sufficiently defined to support a detailed design. If such a design cannot be made, a program manager might infer that additional effort is required to more precisely and firmly define the sometimes elusive user's requirements.

5. The model will probably be adopted by other DoD agencies and defense contractors. Such widespread use should result in large amounts of feedback to RCA concerning the success of the model. Improvements in predictive accuracy should be quickly forthcoming from this feedback.

6. Initially, the model will be based on the actual cost experiences of RCA, a major and experienced developer of both hardware and software

systems. The cost per category of instruction factors will represent values measured by an intensive RCA internal research program. This contrasts to other techniques where the sources of cost factors are frequently not identified.

7. A major advantage of the RCA PRICE model is that it defines in detail the various software cost elements and development phases. With widespread use of the model, these definitions may serve to enhance the flow of software cost information on a common basis.

The Recommendation. There are many problems inhibiting the development of accurate and reliable estimates at ESD. By adopting the RCA PRICE model as a common technique many of these problems can be minimized. Therefore, the early adoption of the RCA PRICE model as the standard ESD software cost estimating methodology is recommended.

Contractor Furnished Cost Information

A second part of the software cost estimation process which can be improved is the management of contractor furnished software cost information. The contractor furnishes such information in his initial cost proposal and in his periodic Cost Performance Reports (CPR). A more uniform policy towards the use of this cost information can significantly improve the software cost estimating environment at ESD.

Cost Proposals. Finding #9 indicated that not all program offices receive data which supports the contractor's initial estimate of software cost. Even where this information is furnished, it sometimes is not sufficient to enable the program office to judge the reasonableness of the estimate. For example, the software cost might be based on an average productivity of eight source instructions per day. However, unless this factor is supported by historical evidence, the program office cannot determine whether the claimed productivity and the resulting cost are reasonable.

Without the ability to completely understand how the contractor arrived at his cost estimate, two problems can occur. The first problem is one of misunderstanding. Difficulties in accurately describing the work to be performed as well as the short time available for the preparation of technical proposals can cause significant technical misunderstandings to arise between the Air Force and the contractor. These misunderstandings can have a negative impact upon both parties and must be avoided.

Finding #10 indicated that these misunderstandings do occur. In one case, contractors' estimate of the size of a software effort ranged from 153,000 instructions to 766,000 instructions. Contractor bids for the same software efforts ranged from a minimum of two-to-one to a maximum of seventeen-to-one for the three procurements studied.

One approach to minimizing these misunderstandings is to have the contractor fully explain the assumptions he made in estimating the cost of software. Significant differences in the size of the effort or in the level of difficulty expected would indicate the need for additional technical discussion between the technical representatives of the Air Force and the contractor. The cost estimate, and the methodology used in deriving it, offer the best vehicle for determining if any gross misunderstandings exist between the Air Force and the contractor. However, to analyze the contractor's estimate of cost, ESD must first require that all cost proposals for software specifically explain the method and assumptions used to make the estimate.

A second potential problem can occur if the cost proposal does not sufficiently detail the software cost estimate. The second problem is the "buy-in" where a contractor knowingly submits an unrealistically low bid for software. As discussed in Chapter II, a buy-in can lead not only to inequitable contract awards, but eventually to additional cost to the Air Force when the true cost of the software becomes apparent. The best method to avoid this problem is to be able to fully analyze the contractor's initial estimate of software costs. If a contractor claims that he will experience a productivity of 20 source instructions per man-day, then the program office can take action to verify the reasonableness of this factor. Audits by the Defense Contract Audit Agency (DCAA) or studies by the Defense Contract Administration Services (DCAS) can be used to verify whether the factor is reasonable based upon the nature of the new work and upon the historical performance of the contractor.

To avoid buy-ins or misunderstandings, it is recommended that all contractor cost proposals for software be required to fully support the software cost estimate. Assumptions as to the size of the effort or as to the expected productivity should be clearly identified and supported by engineering and historical data. With a complete and mutual understanding of the contractor's initial estimate of software cost, a program manager can be better assured that his program is off to a good and solid start.

Cost Performance Reports. Finding #13 explored the cost information submitted by contractors in their periodic Cost Performance Reports. Gross underestimates of software cost were common. These gross errors were always slow to be discovered. Schedules slipped and indirect cost built up. To avoid these problems, a contractor not only needs the ability to produce better initial estimates. He also needs the ability to quickly update this estimate as actual cost and performance data become available to him. In essence the contractor requires a dynamic estimating ability to quickly assess the net impact of actual data. This dynamic ability is required to permit early identification of software problems. The current process is not adequate.

Currently, the contractor simply seems to prepare his initial estimate and continue to work until the actual cost makes it blatantly apparent that the initial estimate was wrong. Major changes in the initial estimate do not occur until it is too late to prevent major schedule slips. Instead of this, a dynamic ability is required so that errors in the estimate are identified quickly.

To obtain this capability, software managers must first understand all of the assumptions which are made in the initial software cost estimate. For example, certain modules are expected to be of a certain length and programmer productivity is expected to be at a certain level. As actual size and productivity data becomes available, the contractor should be required to return to his initial estimate to determine if there were any major errors in his assumptions. If there were major errors, then the entire software estimate may need to be revised. A contractor cannot simply add the additional cost of one module which doubled in size. He must determine if the same doubling in size is possible or probable for the remaining effort.

To aid the contractor in developing a dynamic estimate, the program office should be able to develop their own dynamic estimate. Using the RCA PRICE model, the program office can develop an initial estimate based on the clearly identified assumptions made in the contractor's proposal. As actual data is collected, the program office can re-run the PRICE model to see if the actual data causes any significant changes in the total cost. If significant differences are identified between the program office estimate and the estimate in the Cost Performance Report, quick management action is indicated. The "good times are just around the corner" attitude must be avoided. As pointed out by Finding #13, things never got better in a software development. They always got worse. With this in mind, it is essential that the contractor develop a dynamic estimating capability to quickly utilize the actual cost and performance feedback which is available.

The Recommendation. Contractor furnished software cost information can offer significant improvements to software management if a more uniform and intensive policy is adopted towards the management and utilization of this information. It is therefore recommended that all cost proposals for software be required to fully explain and support the method and assumptions used in estimating software cost. In addition, the estimates provided in the Cost Performance Reports should be dynamic estimates based on the feedback provided from actual cost and performance data.

Future Research Into Software Cost Estimation

As previously stated, the RCA PRICE model may not result in reliable and accurate estimates. While the model does offer promise, it offers no guarantees. Managers at ESD should not be content with simply adopting the RCA PRICE model. Further research into this area is essential and managers should insist that research is continued to better describe the relationships that exist between software cost, software parameters, and weapon system parameters.

However, any further research into this area needs to be conducted with strong direction and according to a definite plan. One only has to look at the many references in this report's bibliography to appreciate that millions of DoD dollars must have been spent in the search for an accurate software cost estimating technique. Yet many of the efforts are repetitive. Most are small efforts with large objectives. Few, if any, build upon the efforts of other researchers. The net result is that despite the expenditure of millions of dollars, the past research efforts have not resulted in the development of a technique which is reliable or accurate. Little if any use is made by today's program managers of yesterday's research into software cost estimating. While the failure of past efforts may have many causes, some seem apparent. Most of the efforts were conducted on an individual basis, sponsored by different agencies, with little correlation or cooperation between the efforts.

To insure that future research efforts do not simply result in more reports and longer bibliographies, research in the area must be strongly guided by a well-founded plan. While this type of research might best be accomplished by a research laboratory such as RADC, the direction of the effort should be centered around the needs of the product divisions such as ESD. Most of the data required for software cost research is currently being collected by those weapon system programs having major software acquisitions. Any research laboratory must rely strongly on this type of data. While the current data might not be uniformly collected, guidance from a laboratory could be used to standardize the type and level of software cost information collected from contractors. With common cost elements and common software development phases, researchers would finally be able to amass enough reliable data to reasonably analyze the relationship between cost and other parameters.

In summary, future research efforts are vitally necessary. However, any further research efforts should be strongly guided by a research plan.

The research laboratory and the product division should cooperate to the fullest extent. The laboratory is dependent upon the product division to provide well-defined software cost data on actual software acquisitions. The product division is likewise dependent upon the laboratory to eventually provide an improved method for estimating the cost of software. Working together jointly, future efforts can provide the information necessary to develop the accurate and reliable software cost estimates required for good software management.

Conclusion

In researching the area of software cost estimation, it was admittedly easier to merely explore how estimates were currently being prepared. The more difficult and more fruitful research remains - to develop a better software cost estimating technique. However, this research has hopefully accomplished its objective of providing managers, researchers, and cost estimators with an increased insight into the problems of the software cost estimating process. This insight may speed the development of better estimation techniques.

However, there is one action which can better improve the estimation of software than the recommendations made by this effort or by any further research effort. To improve software cost estimation, software managers must recognize the vital importance that software cost estimates have in many software management decisions. These estimates are not simply put together to satisfy the requirements of a budget. Instead, the estimate becomes the primary decision criteria in almost every major software management decision. If decision makers want to improve the management of software acquisition, then they must recognize the importance that an accurate and reliable estimate has to the decision process. Once managers recognize this, the increased effort and increased attention will probably result in greater improvement to the software cost estimating process than any other action.

Improving software cost estimates promises to improve software management. However, the reader should remember that cost estimation is only one small facet of the software management environment. If we are to greatly improve our ability to manage software, then we need to continue research in many other critical areas such as software productivity, software configuration control, and software milestones. Cost estimating is only one small part of the management problem.

In conclusion, this research has identified some problems which exist at ESD and may exist at other DoD software acquisition activities. It is hoped that the recommendations resulting from this research effort are favorably considered and adopted by all such activities. Their adoption, coupled with a greater awareness of the problem area and further research, promise to improve the management of software acquisition within DoD.

BIBLIOGRAPHY

1. Air Force Office of Scientific Research. Proceedings of a Symposium on the High Cost of Software Held at the Naval Postgraduate School, Monterey, California, on September 17-19, 1973. Menlo Park, Calif.: Stanford Research Institute, 1973.

2. Aron, J.D. "Estimating Resources For Large Programming Systems" in Conference on Software Engineering Techniques (Rome, Italy 27th to 31st October 1969). Birmingham, England: NATO Science Committee, 1969.

3. Asch, A.; Kelliher, D.W.; Locher, J.P., III; and Connors, T. DoD Weapon Systems Software Acquisition and Management Study, Volume I, MITRE Findings and Recommendations. McLean, Virginia: MITRE Corp., 1975.

4. ______________. DoD Weapon Systems Software Acquisition and Management Study, Volume II, Supporting Material. McLean, Virginia: MITRE Corp., 1975.

5. Aviation Week and Space Technology 104 (April 5, 1976), "Software Improvement Plan Pushed."

6. Boehm, Barry W. Information Processing Requirements For Future Air Force Command and Control Systems, and Some Implications for Software Research and Development. Santa Monica, Calif.: RAND Corp., 1972. (AD 748 084)

7. ____________. Software and Its Impact: A Quantitative Assessment. Santa Monica, Calif.: RAND Corp., 1972.

8. ____________. "The High Cost of Software" in Practical Strategies For Developing Large Software Systems, edited by Ellis Horowitz. Reading, Mass.: Addison-Wesley Publishing Co., 1975.

9. Boehm, Barry W. and Allen C. Haile. Information Processing/Data Automation Implications of Air Force Command and Control Requirements in the 1980s (CCIP-85) Executive Summary. Los Angeles, Calif.: United States Air Force, 1972. (AD 742 292)

10. Brooks, Frederick P., Jr. The Mythical Man-month. Reading, Mass.: Addison-Wesley Publishing Co., 1975.

11. Bucciarelli, M.A. Technical Performance Measurement For Computer Systems. Fort Belvoir, Va.: Defense Systems Management School, 1974.

12. Department of the Army. Costing Methodology Handbook. Washington, D.C.: Comptroller of the Army, 1971. (AD 884 835)

13. Department of the Army, Management Information Systems, Handbook of ADP Resource Estimating Procedures (ADPREP) TB 18-19-1. Washington, D.C.: Headquarters, Department of the Army, 1975.

14. Drabant, T.M. An Introduction To Management of Weapon System Software. Fort Belvoir, Va.: Defense Systems Management School, 1974.

15. Electronic Systems Division (AFSC). Government/Industry Software Workshop Held at ESD on 1-2 October 1974 - Summary Notes. Hanscom AFB, Mass.: ESD, 1975.

16. __________________________________. Air Force ADP Experience Handbook (Pilot Version). Hanscom AFB, Mass.: ESD, 1966. (AD 646 863)

17. Etheridge, D. Computer Software Management. Maxwell Air Force Base, Alabama: Air University, 1974.

18. Farquhar, J.A. A Preliminary Inquiry Into The Software Estimation Process. Santa Monica, Calif.: RAND Corp., 1970.

19. Farr, Leonard. A Description of the Computer Program Implementation Process. Santa Monica, Calif.: System Development Corp., 1963 (AD 299 760)

20. ______________. Quantitative Analysis of Computer Programming Cost Factors: A Progress Report. Santa Monica, Calif.: System Development Corp., 1965. (AD 620 660)

21. Farr, Leonard; LaBolle, Victor; and Willmorth, Norman. Planning Guide for Computer Program Development. Santa Monica, Calif.: System Development Corp., 1965. (AD 465 228)

22. Farr, Leonard and Burt Nanus. Factors that Affect the Cost of Computer Programming. Santa Monica, Calif.: System Development Corp., 1964. (AD 447 520)

23. ______________________. Costs Aspects of Computer Programming for Command and Control. Santa Monica, Calif.: System Development Corp., 1964. (AD 430 259)

24. Farr, Leonard and Henry Zagorski. A Summary of an Analysis of Computer Programming Cost Factors. Santa Monica, Calif.: System Development Corp., 1965. (AD 612 947)

25. _____________________________________. Factors that Affect the Cost of Computer Programming - A Quantitative Analysis. Hanscom AFB, Mass.: ESD, 1964. (AD 607 546)

26. Findley, R.A. Computer Software Development Costs - Predictable or Not? Fort Belvoir, Va.: Defense Systems Management School, 1974.

27. Fleishman, T. Current Results from the Analysis of Cost Data for Computer Programming. Hanscom AFB, Mass.: ESD, 1966. (AD 637 801)

28. Frederic, Brad C. A Provisional Model for Estimating Computer Program Development Costs. Santa Barbara, Calif.: TECOLOTE Research, 1974.

29. Gallagher, Paul F. Project Estimating By Engineering Methods. New York: Hayden Book Company, 1965.

30. Good, Carter V. and Douglas E. Scates. Methods of Research. New York: Appleton-Century-Crofts Inc., 1954.

31. Grimm, Richard. ESD Cost Analysis Division. Wright-Patterson AFB, Ohio. Briefing, 21 May 1976.

32. Headquarters Air Force Systems Command, Draft Proceedings of the Aeronautical Systems Software Workshop. Andrews Air Force Base, Maryland: n.p., April 1974.

33. Heinze, K.; Claussen, N.; and V. LaBolle. Management of Computer Programming for Command and Control - A Survey. Santa Monica, Calif.: System Development Corp., 1963. (AD 415 721)

34. Jacobs, T.O. A Guide for Developing Questionnaire Items. Fort Benning, Georgia: Human Resources Research Organization, 1970. (AD 738 197)

35. Jones, M.V. Estimating Methods and Data Sources Used in Costing Military Systems. Bedford, Mass.: MITRE Corp., 1965. (AD 636 153)

36. LaBolle, V. Critical Management Points. Santa Monica, Calif.: System Development Corp., 1965. (AD 611 867)

37. ________. Estimation of Computer Programming Costs. Santa Monica, Calif.: System Development Corp., 1964. (AD 450 779)

38. ________. Development of Equations for Estimating the Costs of Computer Program Production. Hanscom AFB, Mass.: ESD, 1966. (AD 637 760)

39. Large, Joseph P. Bias in Initial Cost Estimates: How Low Estimates Can Increase the Cost of Acquiring Weapon Systems. Santa Monica, Calif.: RAND Corp., 1974.

40. Levenson, G.S.; Boren, H.E., Jr.; Tihansky, D.P.; and Timson, F. Cost-Estimating Relationships for Aircraft Airframes. Santa Monica, Calif.: RAND Corp., 1971. (AD 891 956)

41. Lientz, Bennet P. Guidelines for the Acquisition of Software Packages. Los Angeles, Calif.: University of California, 1974. (AD 782 477)

42. Malone, John L. Estimating Software Life Cycle Costs. Westlake Village, Calif.: IBM, 1975.

43. Mathis, N.S. Software Milestone Measurement Study. San Diego, Calif.: Naval Electronic Laboratory Center, 1973.

44. Metzger, Philip W. Managing a Programming Project. Englewood Cliffs, New Jersey: Prentice-Hall Inc., 1973.

45. Morin, Lois H. Estimating Computer Programming Projects' Resources. Chapel Hill, N.C.: University of North Carolina at Chapel Hill, 1974.

46. Nanus, Burt. Operations Research Opportunities in Computer Programming Management. Santa Monica, Calif.: System Development Corp., 1964. (AD 450 713)

47. Nelson, E.A. Management Handbook for the Estimation of Computer Programming Costs. Santa Monica, Calif.: System Development Corp., 1965.

48. __________. Methods of Obtaining Estimates of Computer Programming Costs: A Taxonomy. Santa Monica, Calif.: System Development Corp., 1966. (AD 665 478)

49. Nelson, Edward A. and Thomas Fleishman. Cost Reporting for Development of Information Processing Systems. Santa Monica, Calif.: System Development Corp., 1967. (AD 657 793)

50. Nelson, Richard and Alan Sukert. RADC Software Data Acquisition Program. Griffiss AFB, NY: Rome Air Development Center, 1975.

51. Ostwald, Phillip F. Cost Estimating for Engineering and Management. Englewood Cliffs, New Jersey: Cost Estimating for Engineering and Management, 1974.

52. Perry, R.L.; DiSalvo, D.; Hall, G.R.; Harman, A.J.; Levenson, G.S.; Smith, G.K.; and Stucker, J.P. System Acquisition Experience. Santa Monica, Calif.: RAND Corp., 1969.

53. RCA Missile and Surface Radar Division. Computer Program Cost Estimating, Interim Standard. Moorestown, New Jersey: RCA, 1975.

54. Schwartz, Jules I. "Construction of Software:Problems and Practicalities" in Practical Strategies for Developing Large Software Systems, edited by Ellis Horowitz. Reading, Mass.: Addison-Wesley Publishing Co., 1975.

55. Searle, Lloyd V. and Robert L. Henderson. System Engineering Guide for Computer Programs. Hanscom AFB, Mass.: ESD, 1968.

56. Simon, Julian L. Basic Research Methods in Social Science. New York: Random House, 1969.

57. Smith, Ronald L. Structured Programming Series (Vol. XI) - Estimating Software Project Resource Requirements. Gaithersburg, Maryland: IBM, 1975. (AD A016 416)

58. Summers, Robert. Cost Estimates as Predictors of Actual Weapon Costs: A Study of Major Hardware Articles. Santa Monica, Calif.: RAND Corp., 1965.

59. Weinwurm, George F. The Management of Information Processing. Santa Monica, Calif.: System Development Corp., 1965. (AD 619 019)

60. ________________. Data Elements for a Cost Reporting System for Computer Program Development. Hanscom AFB, Mass.: ESD, 1966. (AD 637 804)

61. ________________. On the Management of Computer Programming. Princeton: Auerbach Publishers Inc., 1970.

62. Weinwurm, G.F. and H.J. Zagorski. Research Into the Management of Computer Programming: A Transitional Analysis of Cost Estimation Techniques. Santa Monica, Calif.: System Development Corp., 1965.

63. Willmorth, N.E. and N.S. Mathis. Software Milestone Measurement Study. San Diego, Calif.: Naval Electronics Laboratory Center, 1973. (AD 775 305)

64. Wooldridge, Susan. Software Selection. Philadelphia, PA: Auerbach Publishers Inc., 1973.

65. Wolverton, Ray W. "The Cost of Developing Large Scale Software" in Practical Strategies for Developing Large Software Systems, edited by Ellis Horowitz. Reading, Mass.: Addison-Wesley Publishing Co., 1975.

Appendix A

ACCI/Capt Bourdon/5227/MD/2 Apr 76

ACC 2 April 1976

Software Cost Estimation

DCB	DCM	MCN	OCN	WWE
DCJ	DCY	MCV	OCS	XRI
DCK	FAE	OCD	OCU	YSX
DCL	FAM	OCL	OCW	YWX

1. Software cost estimation is a difficult and critical process. In order to provide some insight into the problems of software cost estimation, Capt. Tom Devenny, an AFIT student, is performing a research effort to describe and study the software cost estimation process at ESD. This research effort should provide program managers a better understanding of software cost estimating and aid software cost estimators in preparing estimates.

2. Your support of this research effort is requested. Capt Devenny will be conducting interviews at ESD from 12 April through 23 April. Request you establish a point of contact for this interview. The point of contact should be the person or persons whom you believe to be most knowledgeable of the software cost estimate(s) made for your program.

3. Request the name(s) of your point of contact be forwarded to ESD/ACCI, Capt Bourdon, by 9 April.

ORIGINAL SIGNED BY

RICHARD W. GRIMM, Major, USAF
Chief, Cost Analysis Division
Comptroller

ACC (File Copy

ACCI
GAB/2Apr76

[illegible] 2 apr

Appendix B

SOFTWARE COST ESTIMATION QUESTIONNAIRE

PRIVACY STATEMENT

In accordance with paragraph 30, AFR 12-35, the following information is provided as required by the Privacy Act of 1974:

a. Authority:

(1) 10 U.S.C., 80-12, Secretary of the Air Force, Powers, Duties, Delegation by Compensation; and/or

(2) EO 93-97, 22 Nov 43, Numbering System for Federal Accounts Relating to Individual Persons; and/or

(3) DOD Instruction 1100.13, 17 Apr 68, Surveys of Department of Defense Personnel; and/or

(4) AFR 178-9, 9 Oct 73, Air Force Military Survey Program.

b. Principal purposes. This survey is being conducted to collect information to be used in research aimed at illuminating and providing inputs to the solution of problems of interest to the Air Force and/or DoD. Specifically, this survey aims to provide managers and software cost estimators greater insight into the area of software cost estimation.

c. Routine uses. The survey data will be converted to information for use in the research of software cost estimation problems. Results of the research, based on the data provided, will be included in a written master's thesis and may also be included in published articles, reports, or texts. Distribution of the results of the research, based on the survey data, whether in written form or presented orally will be unlimited.

d. Participation in this survey is entirely voluntary.

e. No adverse action of any kind may be taken against any individual who elects not to participate in any or all of this survey.

Interviewer: Capt Tom Devenny

Date: ____________________

Time: ____________________

Place: ____________________

1. Briefly describe the software acquisition associated with your program. What is being acquired? When? How?

2. Describe how the program office estimate of software cost was obtained.

3. Briefly describe your educational and work background.

4. How many years have you been involved in systems acquisition? ___# of years

5. How many of these years have you been involved in the acquisition of computer software? ___# of years

6. How many college credit hours of software related courses have you attended? ___# of credit hours

7. Have you ever worked as a programmer? ___Yes ___No
   How long? ___# of years

8. Have you ever directly supervised a group of programmers? ___Yes ___No
   How long? ___# of years

9. Have you ever had any formal training in cost estimation? ___Yes ___No

10. Have you ever estimated the software cost of other projects? ___Yes ___No

11. At the time that the program office made its first formal (e.g. Program Management Plan) estimate of software development costs, how well known were the following factors?

   a. Type of computer (e.g. UYK-7, UNIVAC 1108, etc)

   __Definitely known __Generally known __Slightly known __Unknown

   b. Configuration and types of peripherals

   __Definitely known __Generally known __Slightly known __Unknown

   c. Memory and storage size

   __Definitely known __Generally known __Slightly known __Unknown

   d. Operating system

   __Definitely known __Generally known __Slightly known __Unknown

   e. Compiler and/or assembler

   __Definitely known __Generally known __Slightly known __Unknown

   f. Number and type of interfaces

   __Definitely known __Generally known __Slightly known __Unknown

   g. Number of input message types

   __Definitely known __Generally known __Slightly known __Unknown

h. Number of output message types

__Definitely known __Generally known __Slightly known __Unknown

i. Response time requirements

__Definitely known __Generally known __Slightly known __Unknown

j. Number of Computer Program Configuration Items (CPCIs)

__Definitely known __Generally known __Slightly known __Unknown

12. At the time the contract was awarded, how well known were the following factors?

a. Type of computer (e.g. UYK-7, UNIVAC 1108, etc.)

__Definitely known __Generally known __Slightly known __Unknown

b. Configuration and types of peripherals

__Definitely known __Generally known __Slightly known __Unknown

c. Memory and storage size

__Definitely known __Generally known __Slightly known __Unknown

d. Operating system

__Definitely known __Generally known __Slightly known __Unknown

e. Compiler and/or assembler

__Definitely known __Generally known __Slightly known __Unknown

f. Number and type of interfaces

__Definitely known __Generally known __Slightly known __Unknown

g. Number of input message types

__Definitely known __Generally known __Slightly known __Unknown

h. Number of output message types

__Definitely known __Generally known __Slightly known __Unknown

i. Response time requirements

__Definitely known __Generally known __Slightly known __Unknown

j. Number of Computer Program Configuration Items (CPCIs)

__Definitely known __Generally known __Slightly known __Unknown

13. Are you familiar with the concept of a "management reserve"? __Yes__No

14. Did the program office establish a management reserve for software? ___Yes ___No

15. If yes, was the management reserve left in the program office budget until the contract was essentially complete? ___Yes ___No

16. Do you feel that a software management reserve should be established for each major software acquisition?

__Strongly agree __Agree __No opinion __Disagree __Strongly disagree

17. Do you feel that such a management reserve would be approved during the budget process at HQ AFSC and HQ USAF? ___Yes ___No ___No opinion

18. What size of a management reserve (as a percentage of the estimated software cost) do you feel a program similar to yours should budget? _____%

19. Was an independent cost estimate prepared for software? ___Yes ___No

20. What was the independent estimate? $_________

21. What was the program office estimate at that time? $_________

22. Which estimate do you feel is more accurate? ___SPO ___Independent ___No opinion

23. Did you provide the independent estimator with your estimate of the number of computer instructions? ___Yes ___No

24. Briefly describe the type of information you provided to the independent estimator.

25. Was the accuracy of the SPO's software cost estimate challenged by anyone? ___Yes ___No

26. If yes, by whom? (Please list)

27. Was the accuracy of the contractor's software cost estimate challenged? ___Yes ___No

28. If yes, by whom? (Please list)

29. What was the program office estimate of software cost prior to contract award? $________

30. What was the contractor's estimate of software cost in his proposal? $________

31. How low do you believe the actual software cost might eventually be? $________

32. How high do you believe softrare cost might eventually be? $________

33. What percentage of the contract period is complete? ________%

34. Which of the following contractor tasks were included in the program office estimate of software cost?

a.	Analyze user requirement	___Yes ___No
b.	Prepare system specification	___Yes ___No
c.	Define system interfaces	___Yes ___No
d.	Design the data base	___Yes ___No
e.	Develop program test plans	___Yes ___No
f.	Specify all input and output message formats	___Yes ___No
g.	Design and flow chart each computer program component	___Yes ___No
h.	Write coded program statements	___Yes ___No
i.	Compile and check program code	___Yes ___No
j.	Plan and run functional test of each program	___Yes ___No
k.	Plan and conduct integration test	___Yes ___No
l.	Train user personnel	___Yes ___No
m.	Conduct demonstration test	___Yes ___No

n. Assist in operational shakedown ___Yes ___No

o. Develop software maintenance plan ___Yes ___No

p. Maintain software after delivery ___Yes ___No

35. Did the contractor's cost proposal indicate how the cost of software was estimated? ___Yes ___No

36. If yes, briefly describe the method the contractor used to support his software cost estimate.

37. Was the contract for software awarded competitively? ___Yes ___No

38. What was the lowest software cost contained in any of the contractor proposals (i.e. both successful and unsuccessful bidders)? $________

39. What was the highest software cost contained in any of the contractor proposals (i.e. both successful and unsuccessful bidders)? $________

40. Does/will your contract call for the reporting of cost data? ___Yes ___No

41. Does/will the contractor report the cost of software as a separately identified item? ___Yes ___No

42. Does/will the contractor report the total number of computer instructions? ___Yes ___No

43. Does/will the contractor report the total number of computer instructions for each Computer Program Configuration Item (CPCI)? ___Yes ___No

44. Does/will the contractor report the total number of direct man-hours charged to software? ___Yes ___No

45. Does/will the contractor report the total number of direct man-hours for each CPCI? ___Yes ___No

46. Does/will the contractor report the number of machine hours required for software development and test? ___Yes ___No

47. Does/will the contractor report the number of machine hours required for the development and test of each CPCI? ___Yes ___No

48. What was the contractor's budgeted cost for software at the following points after contract award?

Six months?	$________
Twelve months?	$________
Eighteen months?	$________
Twenty-four months?	$________

49. What was the contractor's actual cost for software at the following points after contract award?

Six months?	$________
Twelve months?	$________
Eighteen months?	$________
Twenty-four months?	$________

50. What was the percent complete reported for software at the following points after contract award?

Six months?	$________
Twelve months?	$________
Eighteen months?	$________
Twenty-four months?	$________

51. What was the program office estimated cost of software? $________

52. What was the software cost estimate in the contractor's proposal? $________

53. What was the software cost estimate at the System Design Review? $________

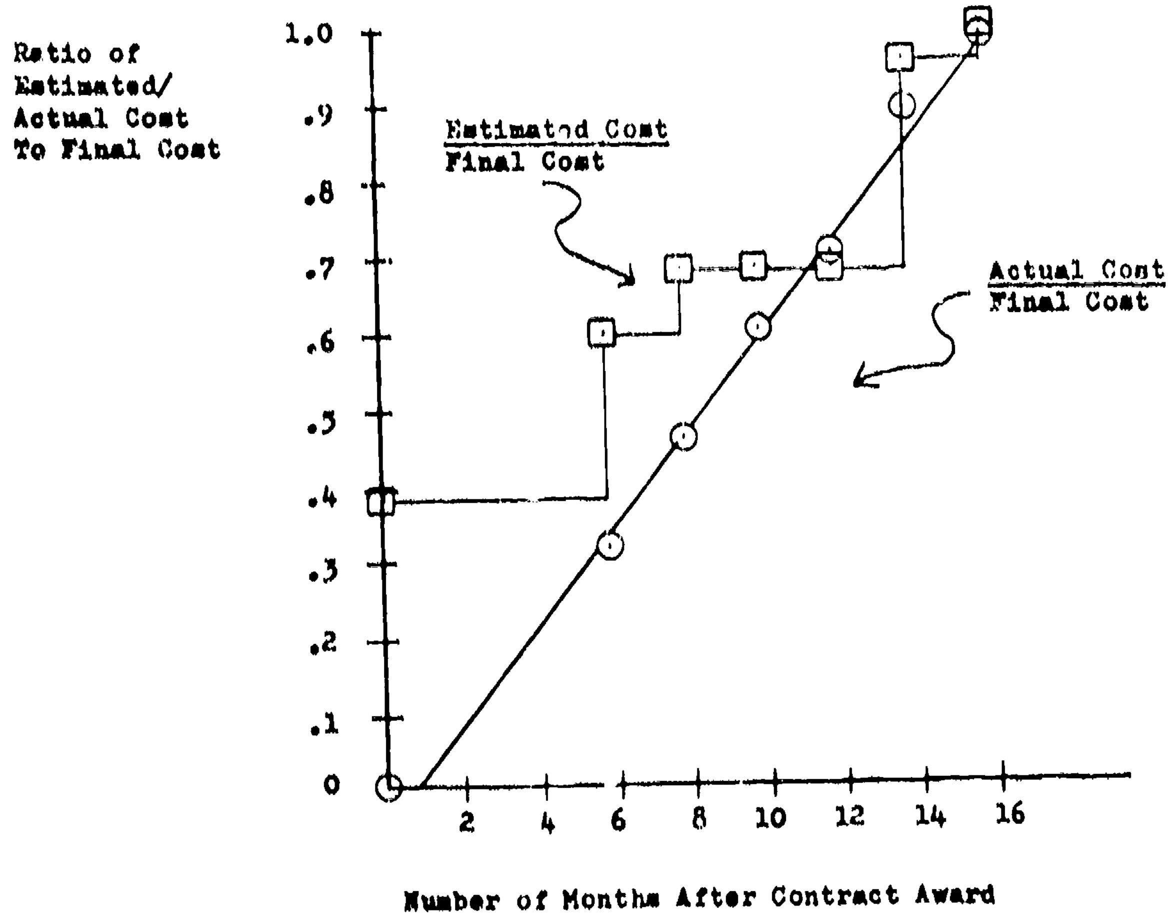
54. How many months after contract award was the System Design Review held? ___# of months

55. What was the software cost estimate at the Preliminary Design Review? $________

56. How many months after contract award was the Preliminary Design Review held? ___# of months

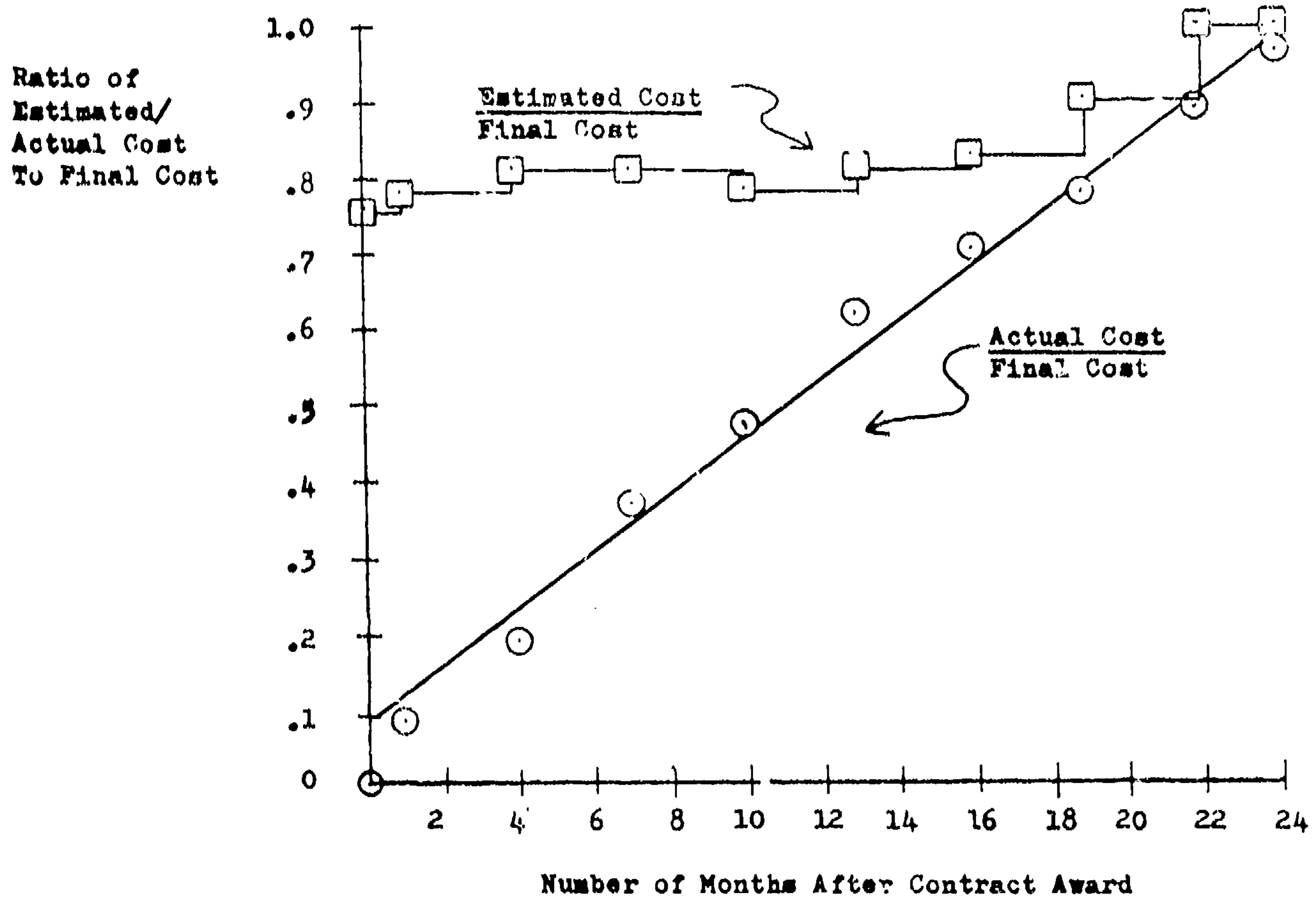
57. What was the software cost estimate at the Critical Design Review?	$________
58. How many months after contract award was the Critical Design Review held?	___# of months
59. What is the current software cost estimate?	$________
60. How many months have elapsed since contract award?	___# of months
61. How many months remain until contract completion?	___# of months

Appendix C



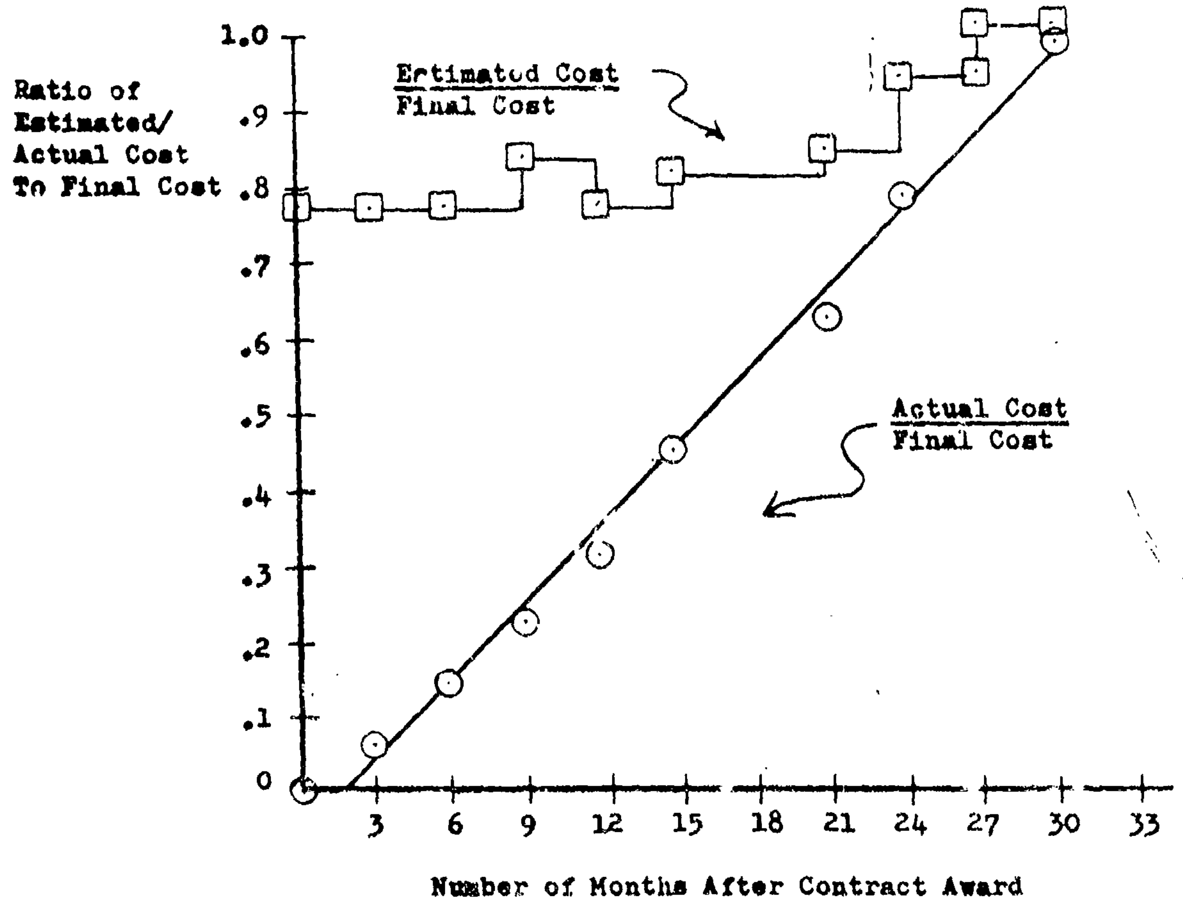
PROGRAM B - Cost Performance Data

Figure 7



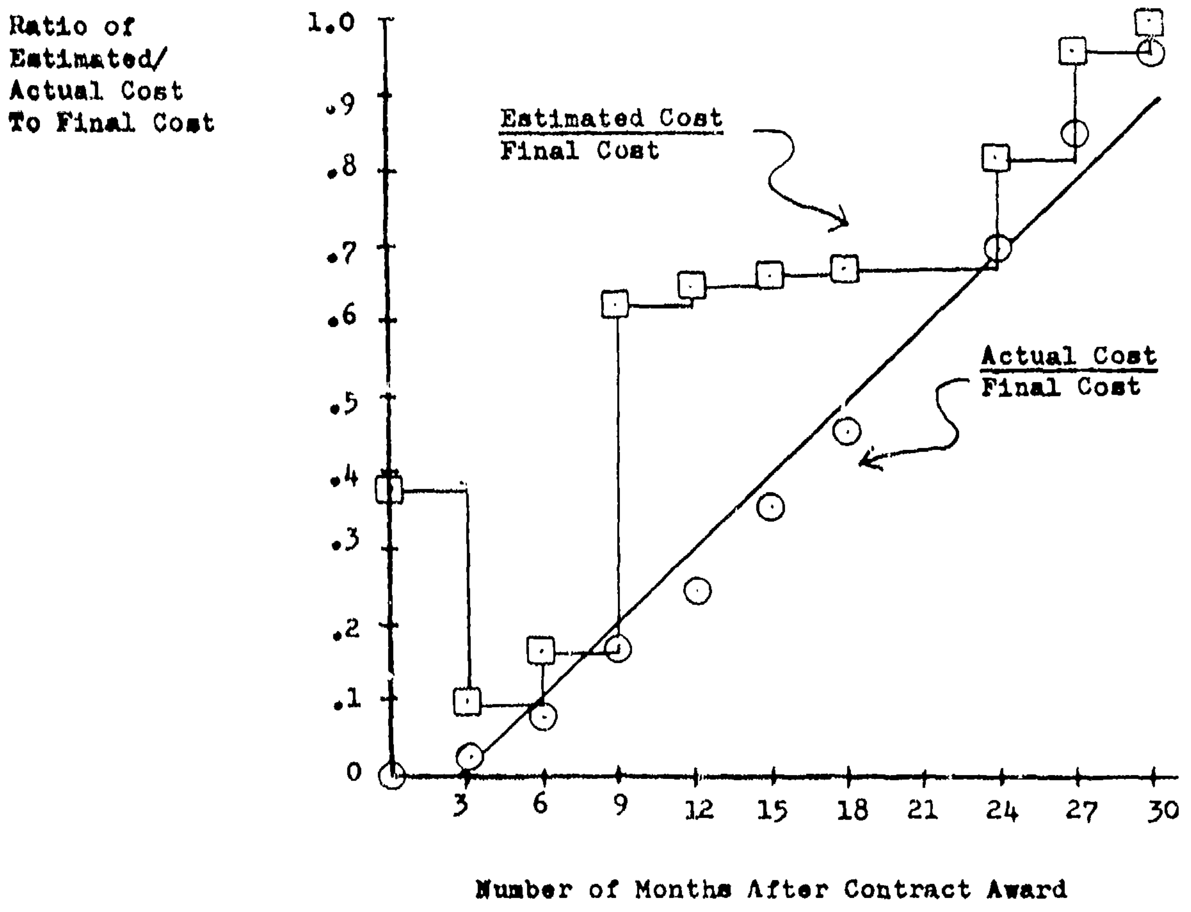
PROGRAM C - Cost Performance Data

Figure 8



PROGRAM D - Cost Performance Data

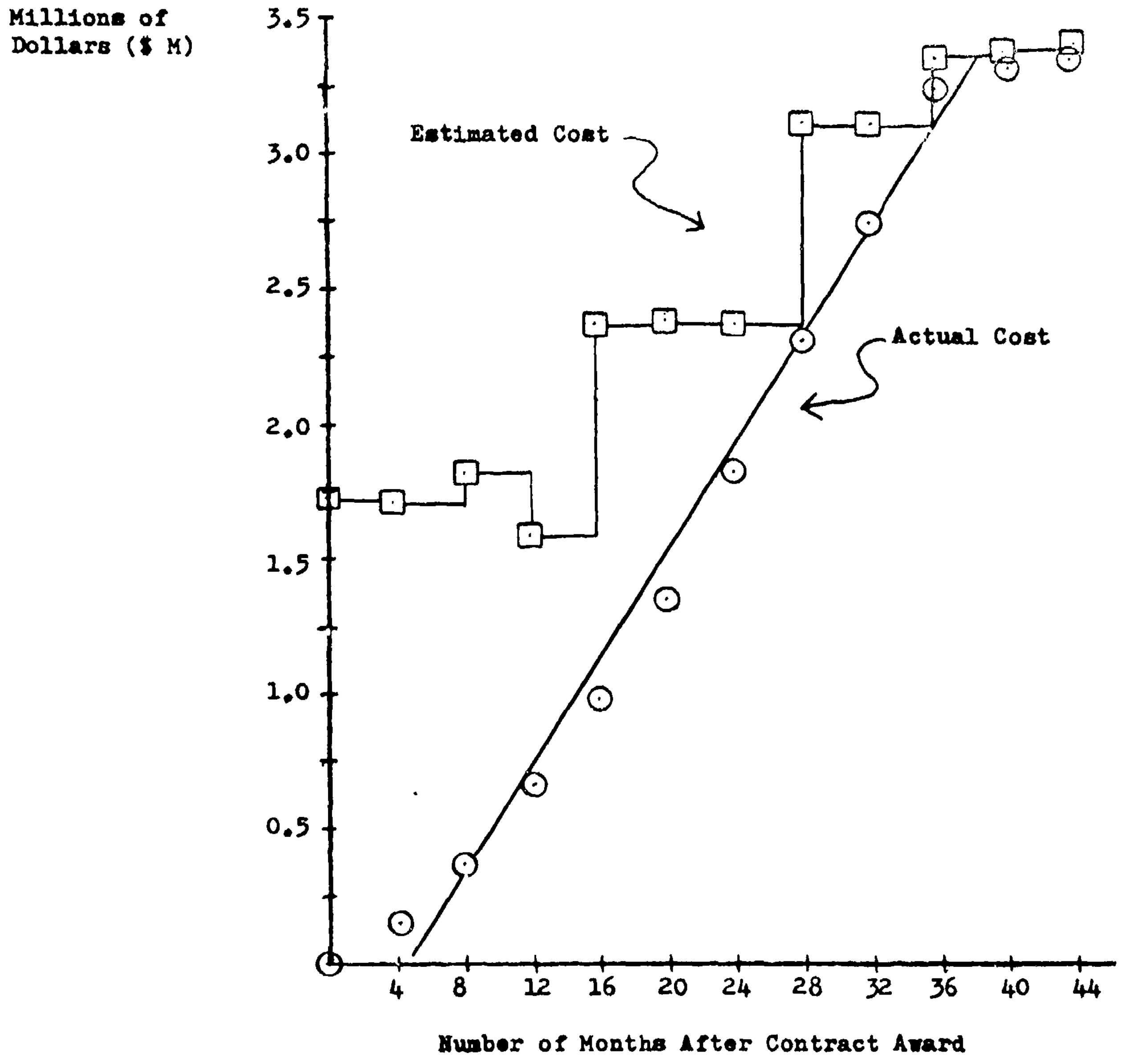
Figure 9



PROGRAM 3 - Cost Performance Data

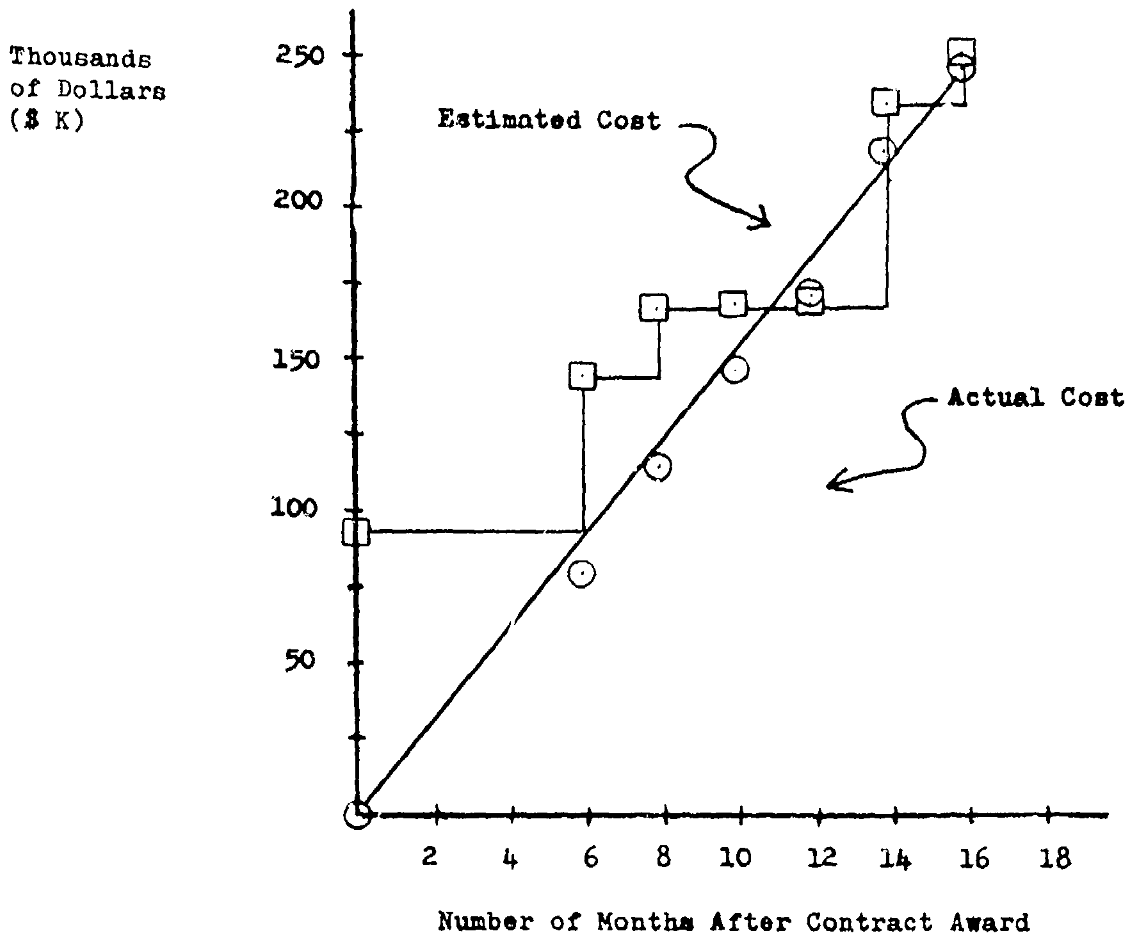
Figure 10

Appendix D



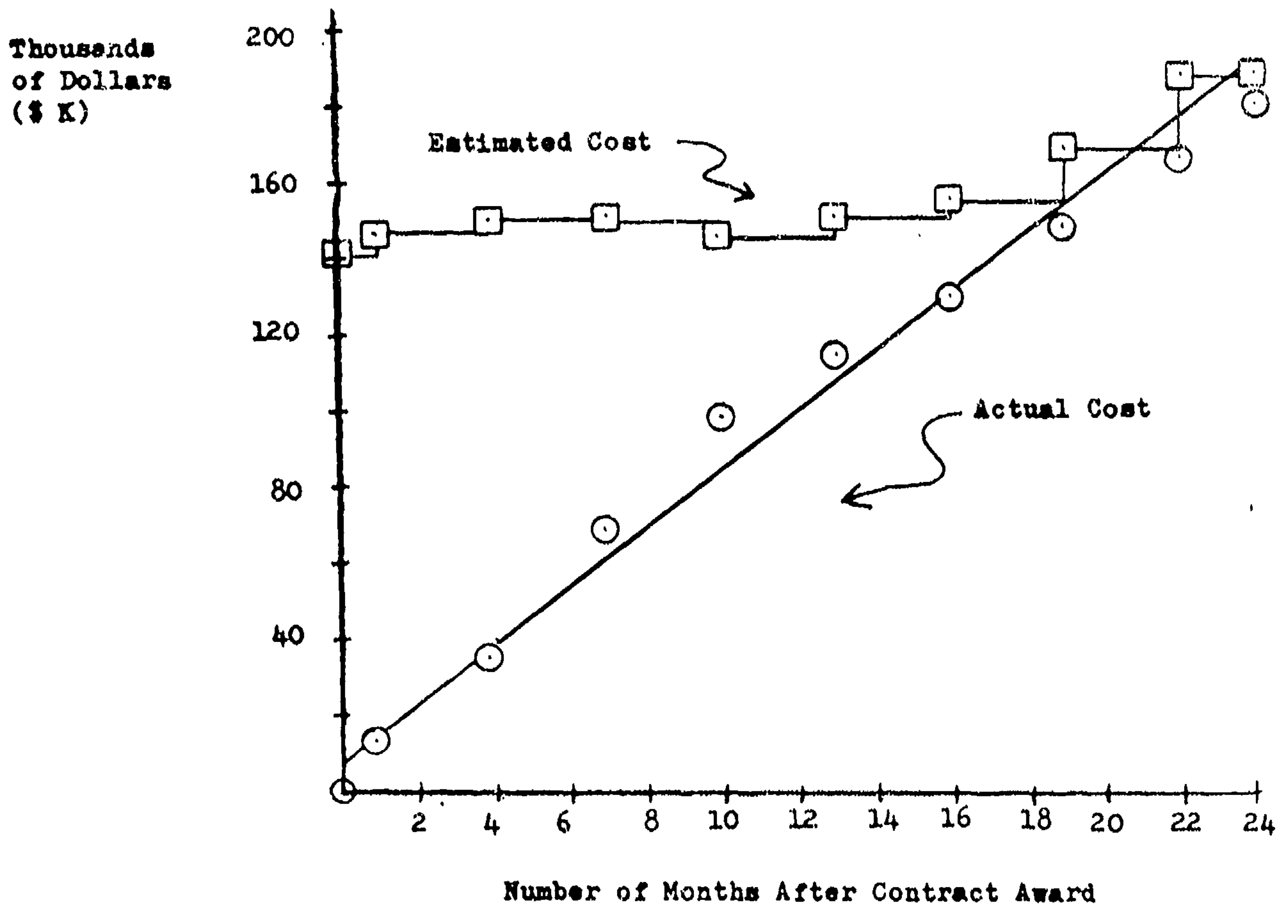
PROGRAM A - Cost Performance Data

Figure 11



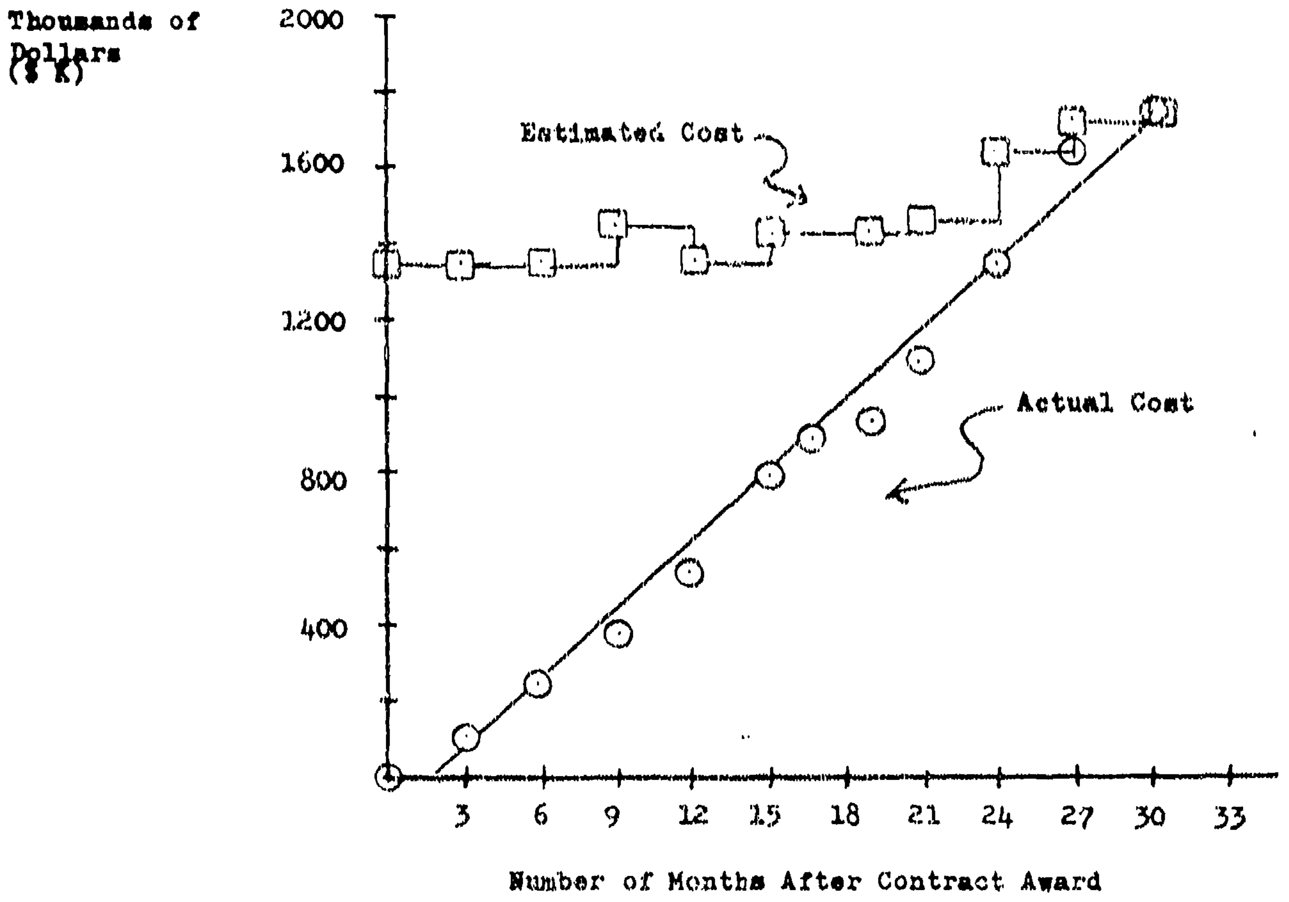
PROGRAM B - Cost Performance Data

Figure 12



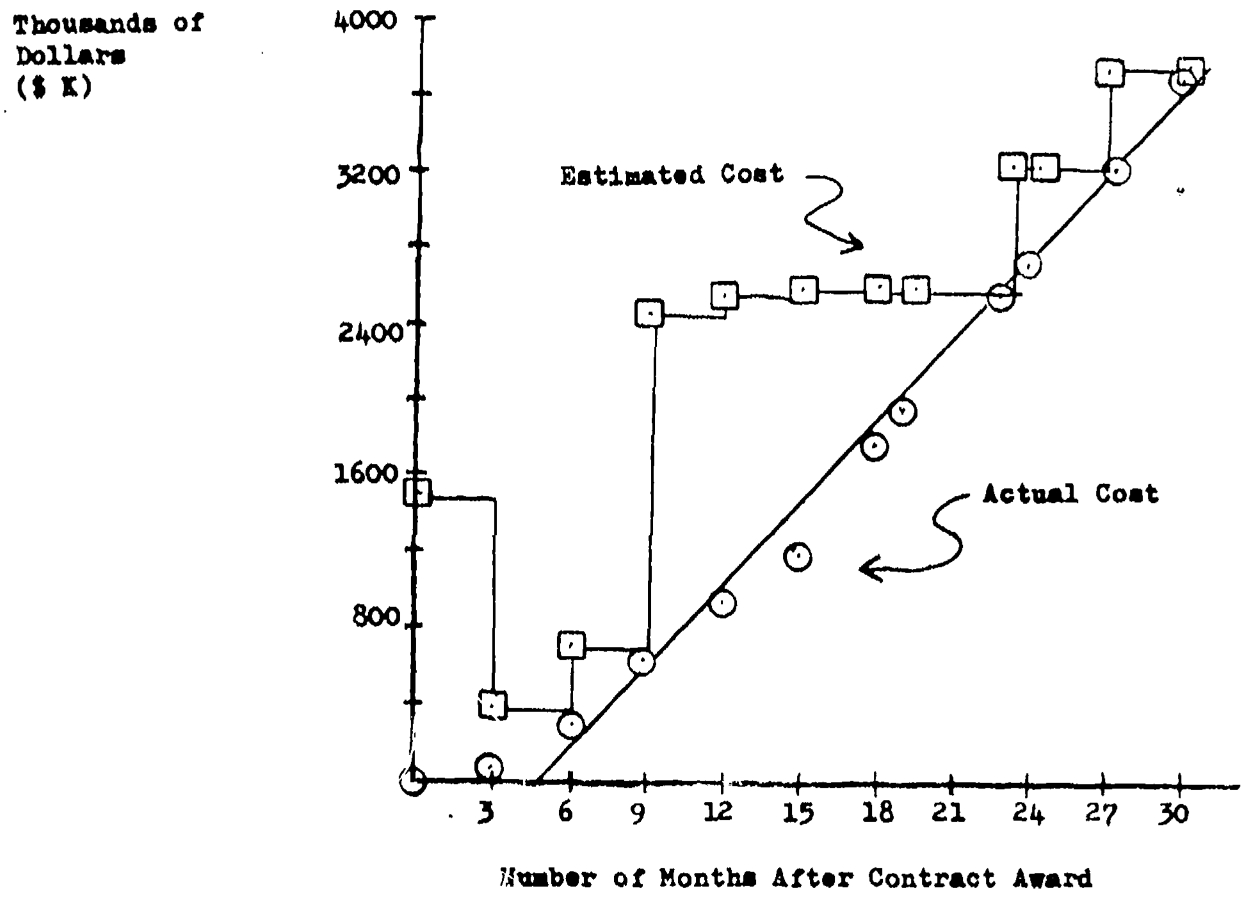
PROGRAM C - Cost Performance Data

Figure 13



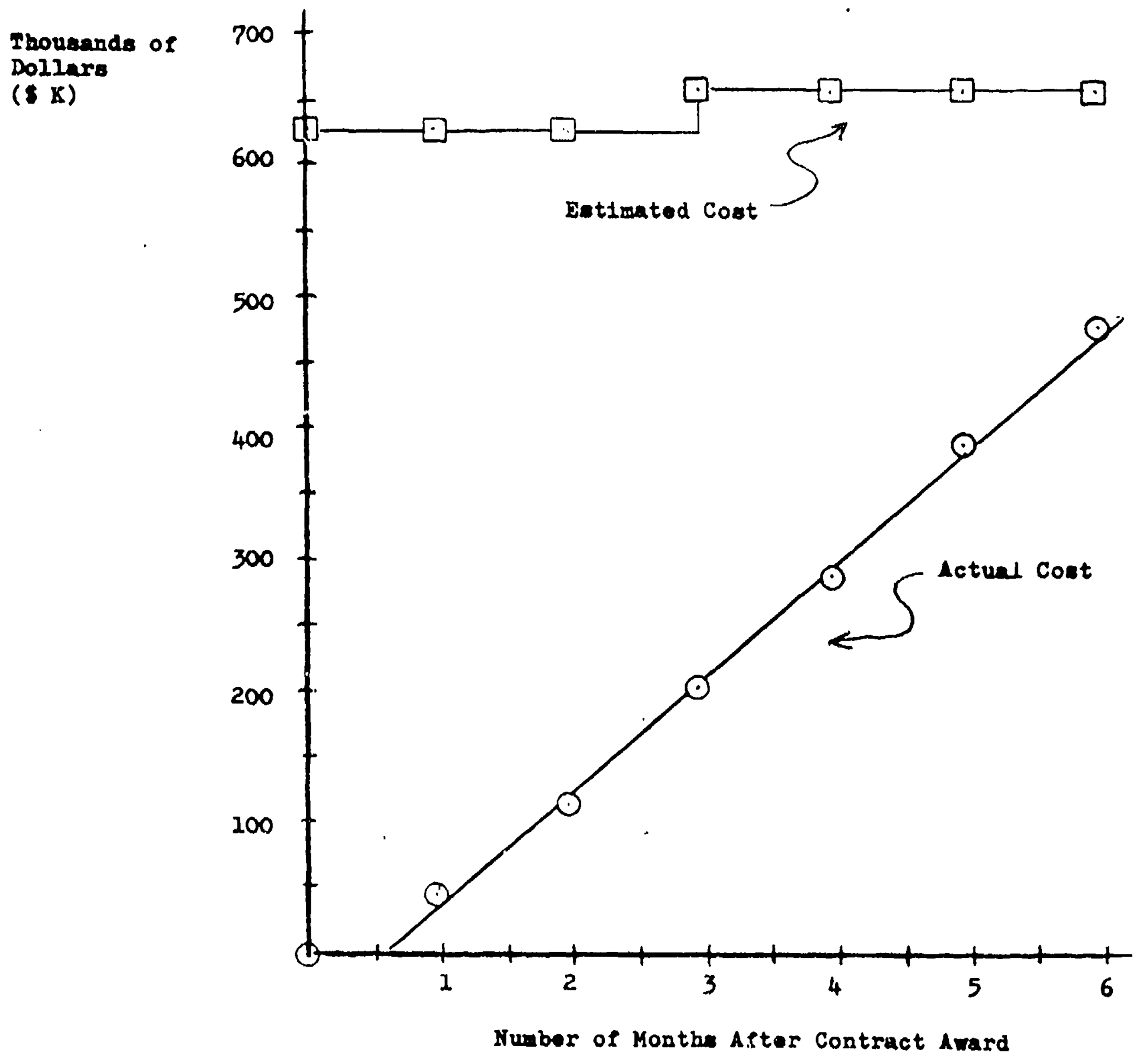
PROGRAM D - Cost Performance Data

Figure 14



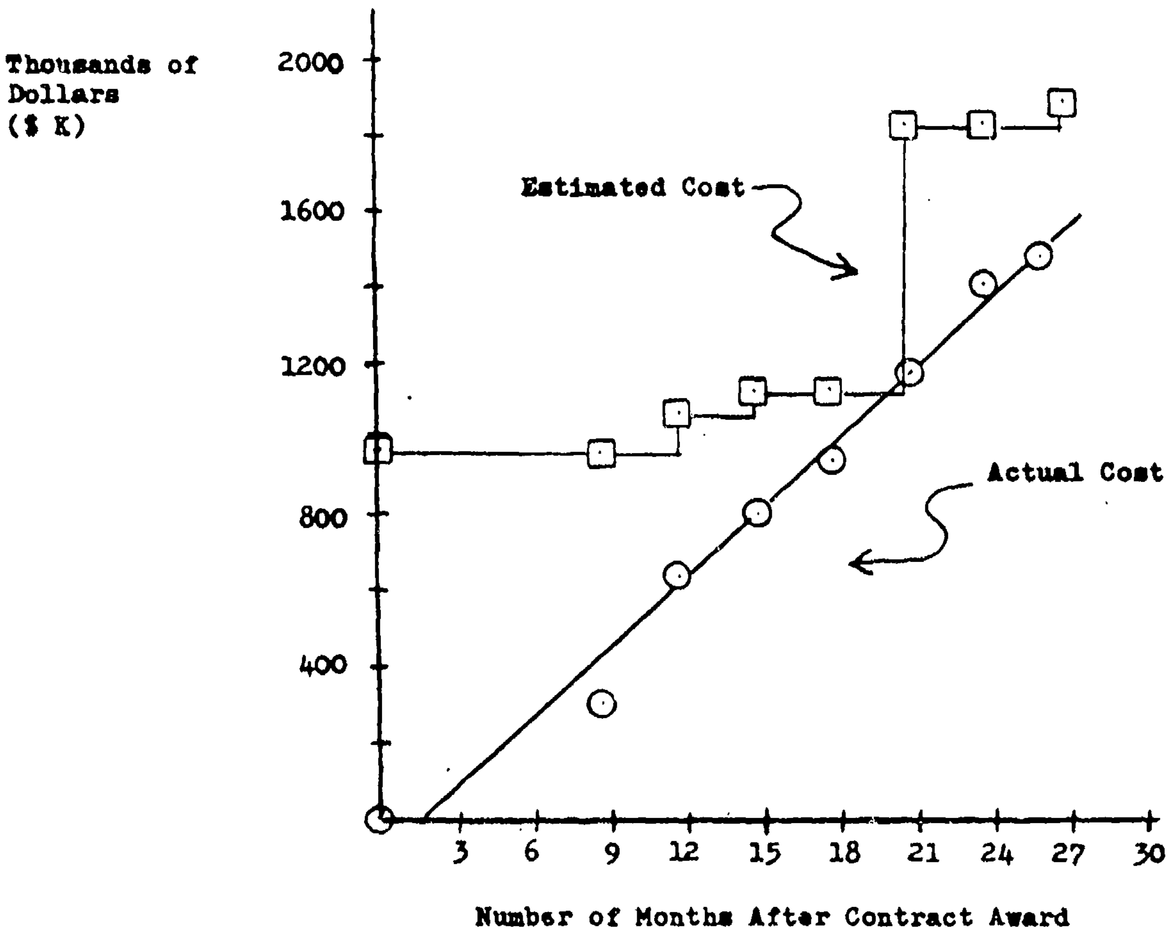
PROGRAM E - Cost Performance Data

Figure 15



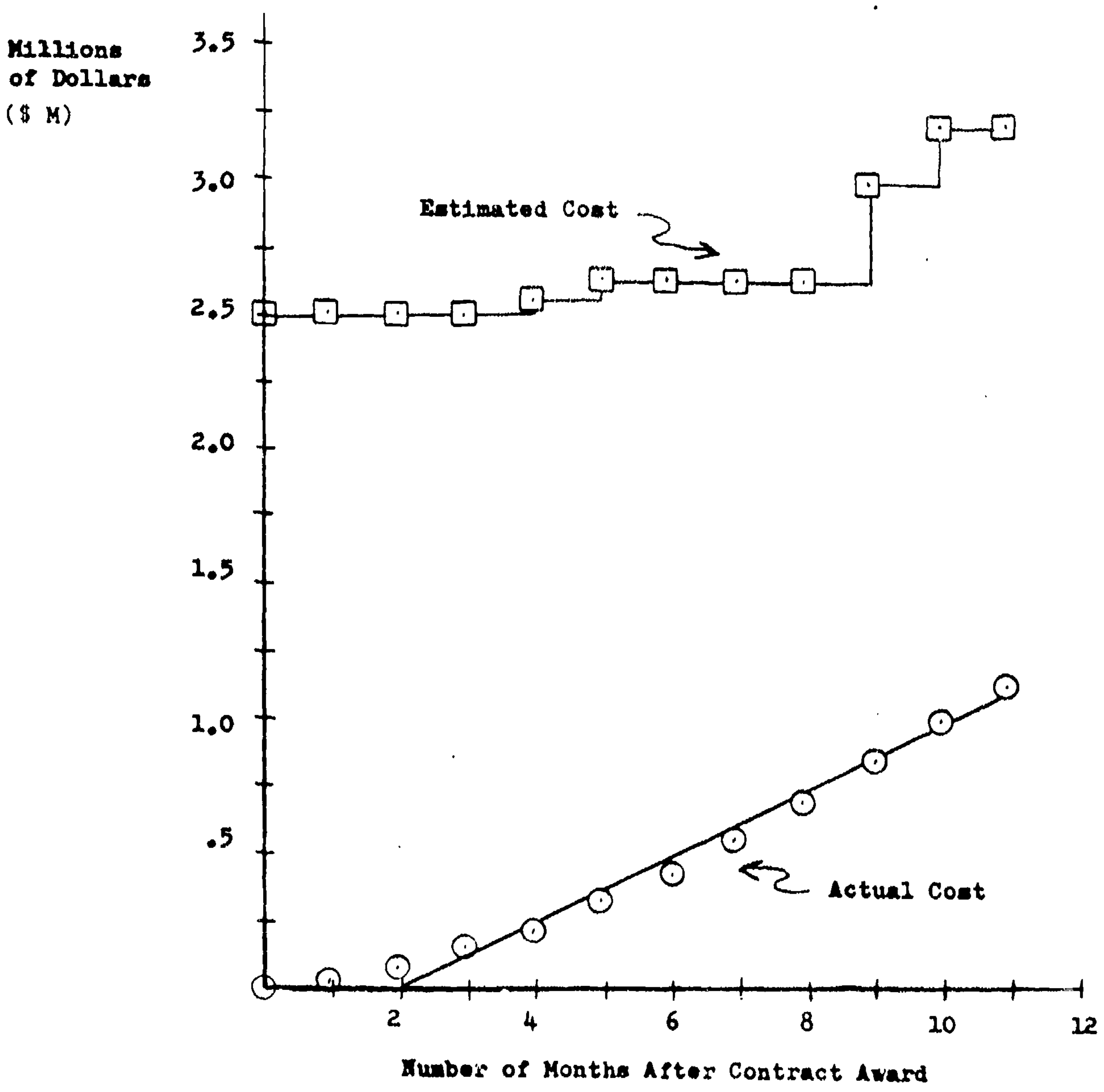
PROGRAM G - Cost Performance Data

Figure 16



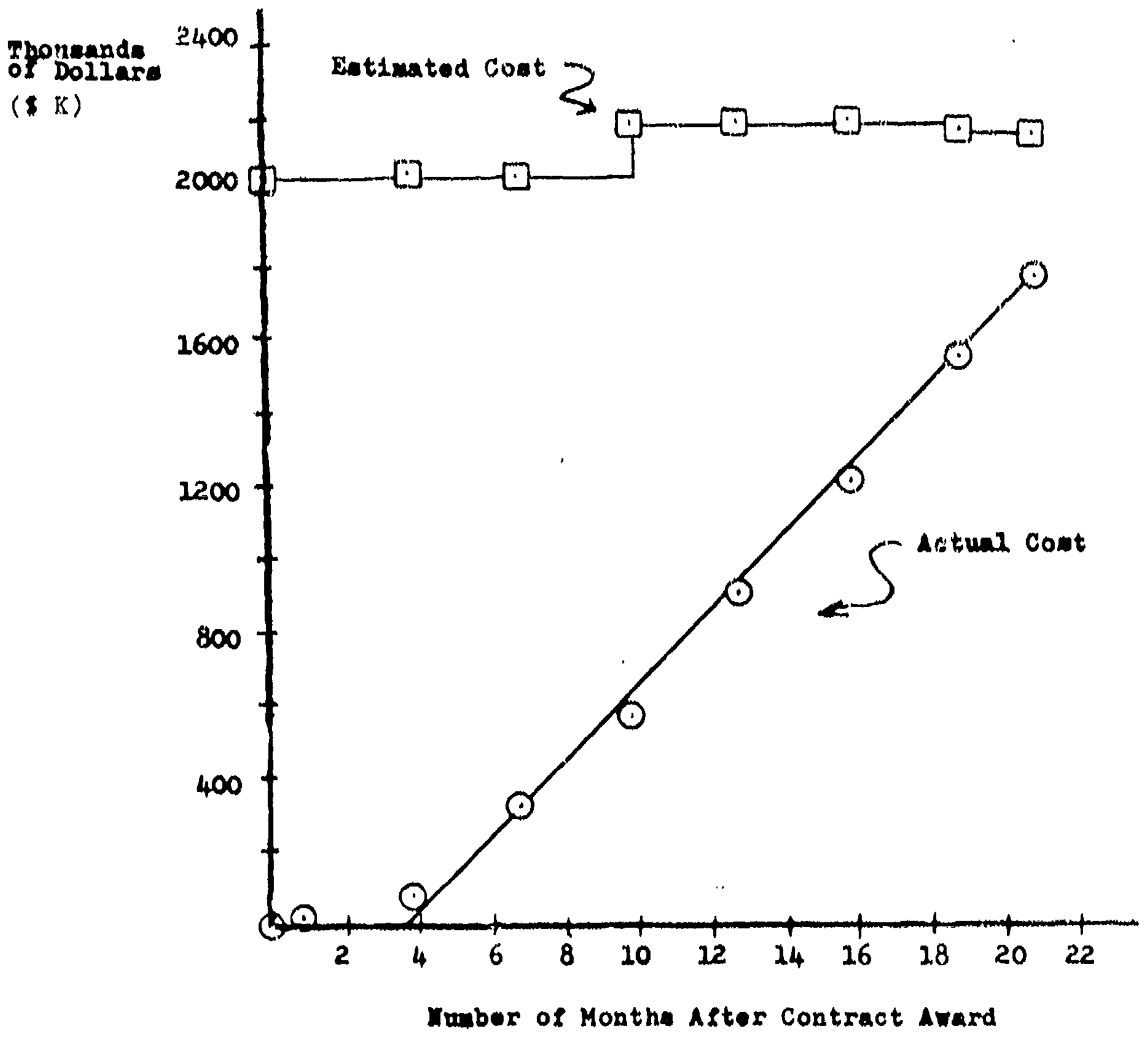
<u>PROGRAM H - Cost Performance Data</u>

<u>Figure 17</u>



<u>PROGRAM I - Cost Performance Data</u>

<u>Figure 18</u>



PROGRAM J - Cost Performance Data

Figure 19

VITA

Captain Thomas Joseph Devenny was born in New York City on July 7, 1945. He attended Manhattan College receiving a Bachelor of Science degree in Electrical Engineering. Commissioned through Officer Training School, he was first assigned as a project engineer/manager at the Rome Air Development Center, Griffiss AFB, New York and was responsible for the development and installation of three unique data processing systems for USAFE, FTD, and USAFSS. Through a combination of luck and dedicated contractors, these systems were successful and are still in active use after many years.

After the tour at the AFSC laboratory, Capt Devenny was assigned to the DCS/Systems of Headquarters, Air Force Systems Command as a system officer. Responsible for the management and budgeting of numerous RADC and ESD projects, he also participated in studies concerning the reorganization of AFSC and the acquisition of command, control, and communications systems.

Capt Devenny is currently assigned to the School of Engineering, AFIT, where he is pursuing a master's degree in Systems Management.

Capt Devenny is married to the former Miss Betty Chomin of Rome, New York. They have two children, Kimberly and Michael.

Permanent address: 433 East 80 Street
New York, New York 10021

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (*When Data Entered*)

REPORT DOCUMENTATION PAGE | **READ INSTRUCTIONS BEFORE COMPLETING FORM**

1. REPORT NUMBER GSM/SM/76S-4.	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (*and Subtitle*) AN EXPLORATORY STUDY OF SOFTWARE COST ESTIMATING AT THE ELECTRONIC SYSTEMS DIVISION		5. TYPE OF REPORT & PERIOD COVERED M.S. THESIS 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) THOMAS J. DEVENNY CAPTAIN USAF		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS AIR FORCE INSTITUTE OF TECHNOLOGY (EN) WRIGHT-PATTERSON AFB, OHIO 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS AFIT
11. CONTROLLING OFFICE NAME AND ADDRESS AIR FORCE INSTITUTE OF TECHNOLOGY (EN) WRIGHT-PATTERSON AFB, OHIO 45433		12. REPORT DATE JULY 1976 13. NUMBER OF PAGES 152
14. MONITORING AGENCY NAME & ADDRESS(*if different from Controlling Office*)		15. SECURITY CLASS. (*of this report*) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (*of this Report*)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (*of the abstract entered in Block 20, if different from Report*)

18. SUPPLEMENTARY NOTES

Approved for public release; IAW AFR 190-17

[illegible]

Jerral F. Guess, Capt, USAF
Director of Information

19. KEY WORDS (*Continue on reverse side if necessary and identify by block number*)

Computer programming, computer software, cost estimating, software management, software cost estimating.

20. ABSTRACT (*Continue on reverse side if necessary and identify by block number*)

The estimate of software development cost is the key piece of information in many software management decisions. No technique exists which can consistently produce the reliable and accurate cost estimates which managers need. This thesis research effort explored the software cost estimating process at the Electronic Systems Division of the Air Force Systems Command. The purpose of the research was to provide managers, researchers, and cost estimators with a better insight into the cost estimating process.

Data were gathered from 16 major software acquisitions at ESD using both

over

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (*When Data Entered*)

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

a structured interview and contractor furnished Cost Performance Reports. The research findings identified some major problems which are currently inhibiting the development of accurate and reliable software cost estimates.

To reduce these problems, recommendations are made to adopt a common cost estimating technique and to modify the use of contractor furnished software cost information. While the research was limited to ESD, the research findings and the recommendations may be applicable to other DoD software acquisition agencies.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)